Universidad de Alcalá

Escuela Politécnica Superior

Máster Universitario en Ingeniería Industrial

Trabajo Fin de Máster

"Localización y mapeado simultáneos para vehículos autónomos utilizando Lidar 3D"

Autora: Rocío Ribalda Fernández

Tutora: Elena López Guillén

2020

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

MÁSTER EN INGENIERÍA INDUSTRIAL

Trabajo Fin de Máster

"Localización y mapeado simultáneos para vehículos autónomos utilizando Lidar 3D"

Autor: Rocío Ribalda Fernández

Tutor/es: Elena López Guillén

TRIBUNAL:

Presidente: Luis Miguel Bergasa Pascual

Vocal 1º: Saturnino Maldonado Bascón

Vocal 2º: Elena López Guillén

Calificación: ……

Fecha: 27/10/2020

"Esfuérzate, no para ser un éxito, sino más bien para ser de valor."

Albert Einstein

Agradecimientos

La finalización de este Trabajo Fin de Máster supone el fin de una etapa, llena de obstáculos y dudas, pero en la que el sacrificio ha merecido la pena.

En primer lugar, me gustaría agradecer la oportunidad brindada por mi tutor Elena López Guillén de realizar un trabajo realmente interesante, que me ha ayudado a afianzar muchos de los conocimientos aprendidos durante mi estancia en la universidad. También agradecer al grupo de investigación Robesafe la acogida y la ayuda durante todos estos meses.

A mis padres, por inculcarme desde pequeña que con esfuerzo y constancia todo se consigue. Sin sus consejos y su cariño hubiera sido imposible llegar hasta aquí.

A mis abuelos, soy lo que soy en parte gracias a vosotros, nunca tendré las suficientes palabras para agradeceros todo lo que habéis hecho por mí.

A mi hermano, por hacerme reír cuando más lo necesitaba y confiar plenamente en mí.

A mis amigos, por apoyarme y sacarme siempre una sonrisa en los momentos de mayor estrés.

A Javier, por ayudarme y creer en mí cuando ni siquiera yo lo hacía, por ser un pilar fundamental en mi vida, por impulsarme a ser mejor, porque sin ti nada hubiera sido igual.

Índice general

Resumen 1

Abstract 3

Resumen extendido 5

Capítulo 1 Introducción. 7

1.1. Entorno de trabajo y motivación 7

1.2. Objetivos del trabajo 9

1.3. Estructura del trabajo 9

Capítulo 2. Estado del arte 11

2.1. Introducción 11

2.2. El problema del SLAM 11

2.3. Sistemas del SLAM basados en Lidar 3D 14

2.4. Implementaciones para ROS 15

2.4.1. Mrpt Localization 16

2.4.2. Gmapping 17

2.4.3. Hector Slam 19

2.4.4. MCL_3dl 20

2.4.5. Google Cartographer 21

2.4.6. Loam SLAM 22

2.4.7. Hdl_Graph_SLAM 23

Capítulo 3. Herramientas utilizadas. 25

3.1. Simulador CARLA 25

3.2. Robot Operating System 28

3.2.1. Conceptos básicos de ROS 30

3.2.2. Visualizador Rviz 31

3.3. Paquetes de ROS 33

3.3.1. CARLA-ROS-bridge 33
3.3.2. Cartographer ROS 35
3.3.3. Loam: Lidar Odometry and Mapping in Real-time 37
3.3.4. Hdl_graph_slam 39
3.3.5. Vehículo Teach4AgeCar 41
Capítulo 4. Desarrollo 49
4.1. Lidar 3D SLAM con Google Cartographer 49
4.1.1. Algoritmo 49
4.1.2. Topics y Parámetros 53
4.1.3. Resultados con CARLA Simulator 58
4.2. Lidar 3D SLAM con Lidar Odometry and Mapping (LOAM) 60
4.2.1. Algoritmo 60
4.2.2. Topics y Parámetros 62
4.2.3. Resultados con CARLA Simulator 64
4.3. Lidar 3D SLAM con Hdl_Graph_SLAM 66
4.3.1. Algoritmo 66
4.3.2. Topics y Parámetros 68
4.3.3. Resultados con CARLA Simulator 72
Capítulo 5. Resultados 75
5.1. Comparativa de los métodos de SLAM utilizando el simulador 75
5.1.1. LOAM SLAM 76
5.1.2. Hdl_Graph_Slam 77
5.2. Resultados con el vehículo autónomo en entorno real 80
5.2.1. Google Cartographer 81
5.2.2. LOAM SLAM 82
5.2.3. Hdl_Graph_SLAM 84
5.3. Método seleccionado 85

5.3.1. Odometría LOAM VS Odometría Encoders 85
5.3.2. Fusión con GPS en filtro de Kalman extendido 86
Capítulo 6. Conclusiones y trabajos futuros. 91
Anexo I. Manual de usuario 93
I.I Simulador Carla 93
I.II Google Cartographer 95
I.III LOAM SLAM 96
I.IV Hdl_Graph_SLAM 96
Anexo II. Presupuesto 99
II.I Coste Hardware 99
II.II Coste Software 99
II.III Costes de personal 100
II.IV Costes de ejecución totales 100
II.V Gastos generales y beneficio industrial 100
II.VI Importe total del presupuesto 101
Anexo III. Pliego de condiciones 103
III.I Software empleado 103
III.II Hardware empleado 103
Bibliografía 105

Tabla de ilustraciones

Ilustración 1 Vehículo autónomo grupo Robesafe, Universidad de Alcalá 8
Ilustración 2 Error lazo de control aplicando técnicas de SLAM 13
Ilustración 3 Vehículo autónomo de Stanford e histograma 2D en tiempo real 14
Ilustración 4 Fusión sensorial aplicada en BMW [9] para localización. 15
Ilustración 5 Mrpt Localization 16
Ilustración 6 Split-and-merge algorithm. Gmapping 18
Ilustración 7 Gmapping SLAM 18
Ilustración 8 Mapa generado por Hector SLAM 19
Ilustración 9 MCL_3dl 21
Ilustración 10 Mapas urbanos ante cambios de las condiciones climáticas. CARLA Simulator 26
Ilustración 11 Mundo CARLA simulator 27
Ilustración 12 Modalidades de sensores CARLA simulator 28
Ilustración 13 Programación empleando plataformas robóticas 29
Ilustración 14 Comunicación ROS Master 30
Ilustración 15 Esquema Topics ROS 31
Ilustración 16 Visualizador RViz 32
Ilustración 17 Árbol de transformadas RViz 33
Ilustración 18 Visualizador CARLA ROS-bridge 34
Ilustración 19 Deutsches Musesum Google Cartographer 36
Ilustración 20 Algoritmo LOAM 38
Ilustración 21 Diagrama de bloques del sistema software LOAM 39
Ilustración 22 Algoritmo Hdl_Graph_SLAM 39
Ilustración 23 Esquema global funcionamiento Hdl_Graph_SLAM 40
Ilustración 24 Vehículo autónomo grupo Robesafe Universidad de Alcalá 41
Ilustración 25 Arquitectura de navegación autónoma 42
Ilustración 26 Medidor de distancia láser 43
Ilustración 27 Reconstrucción 3D del entorno con láser 3D 44
Ilustración 28 Sistema láser 44
Ilustración 29 VelodyneLidarPuck (VLP16) 45
Ilustración 30 Funcionamiento sistema GPS 45
Ilustración 31 Sistema de sensores del proyecto Teach4AgeCar 46

Ilustración 32 Orientación de los sensores Vehículo autónomo Robesafe 46
Ilustración 33 Esquema del módulo de Localización 47
Ilustración 34 Esquema básico Local SLAM Google Cartographer 50
Ilustración 35 Esquema básico TFM Cartographer 53
Ilustración 36 Árbol de transformadas Cartographer 56
Ilustración 37 Recorrido CARLA Simulator 58
Ilustración 38 Recorrido Google Cartographer 59
Ilustración 39 Recorrido en perspectiva Google Cartographer 60
Ilustración 40 Puntos característicos LOAM 61
Ilustración 41 Desarrollo de un barrido láser, Google Cartographer 61
Ilustración 42 Esquema básico TFM LOAM 63
Ilustración 43 Árbol de transformadas LOAM 64
Ilustración 44 Recorrido CARLA Simulator 64
Ilustración 45 Recorrido mundo CARLA aplicando LOAM SLAM 65
Ilustración 46 Recorrido mundo Carla aplicando LOAM SLAM vista de perspectiva 66
Ilustración 47 Estructura gráfico de posición 67
Ilustración 48 Ejemplo detección plano del suelo 67
Ilustración 49 Esquema básico TFM Hdl_graph_slam 69
Ilustración 50 Coordenadas GPS Hdl_Graph_SLAM 71
Ilustración 51 Árbol de transformadas Hdl_Graph_SLAM 71
Ilustración 52 Recorrido CARLA Simulator 72
Ilustración 53 Recorrido mundo CARLA aplicando Hdl_Graph_SLAM 72
Ilustración 54 Recorrido mundo CARLA aplicando Hdl_Graph_SLAM vista de perspectiva 73
Ilustración 55 Comparativa Ground Truth CARLA VS LOAM Localization 76
Ilustración 56 Comparativa Ground Truth CARLA VS Hdl Localization 78
Ilustración 57 Odometría láser Hdl_Graph_SLAM RViz 78
Ilustración 58 Comparativa Ground Truth CARLA VS Hdl_Graph_SLAM Localization 79
Ilustración 59 Recorrido Real Vehículo Autónomo, LOAM SLAM 81
Ilustración 60 Recorrido entorno real aplicando LOAM SLAM 82
Ilustración 61 Recorrido entorno real vista en perspectiva, LOAM SLAM 82
Ilustración 62 Comparativa localización actual vehículo real VS LOAM Localization 83
Ilustración 63 Recorrido en entorno real aplicando Hdl_Graph_SLAM 84
Ilustración 64 Recorrido entorno real vista en perspectiva, Hdl_Graph_SLAM 84

Ilustración 65 Resultado Techs4AgeCar Odometría Encoders VS Odometría LOAM 86
Ilustración 66 Esquema Filtro EKF con LOAM 87
Ilustración 67 Transformación de los sistemas de coordenadas 87
Ilustración 68 Resultado Techs4AgeCar LOAM con EKF 88
Ilustración 69 Zoom Resultado Techs4AgeCar LOAM con EKF 88

Resumen

El objetivo de este Trabajo Fin de Máster es realizar un estudio de las diferentes técnicas de localización y mapeado simultaneo (Simultaneous Localization and Mapping-SLAM) que existen actualmente, para poder implementarlas en el coche autónomo del grupo de investigación RobeSafe de la Universidad de Alcalá. Estas técnicas pretenden conseguir que, al colocar un vehículo en una posición desconocida, este sea capaz de construir incrementalmente un mapa del entorno y que al mismo tiempo lo use para detectar cuál es su localización dentro de dicho mapa.

La complejidad se halla en encontrar cuál es la técnica de SLAM más apropiada para la aplicación deseada, comparando entre los métodos ya existentes y adaptándolos a las características del proyecto Techs4AgeCar del grupo Robesafe. Una de las principales especificaciones del trabajo es que el sensor principal utilizado sea un Lidar 3D, de manera que el sistema de SLAM obtenido pueda integrarse al módulo de localización del vehículo autónomo del citado proyecto, que ya integra otros sensores como odometría, GPS y sistemas inerciales.

Palabras clave: SLAM, láser, navegación autónoma, ROS, mapeado, localización, CARLA, CARLA-ROS Bridge, Cartographer, Blam, Hdl_graph_slam, Loam.

Abstract

The main aim of this project is to analyze the different technics of Simultaneous Localization and Mapping that currently exist, to be able to implement them in the autonomous car of the Robesafe research group. These technics pretend to achieve that when the car is in an unknown position the vehicle will be able to incrementally build a map of the environment and at the same use it to detect its localization into the map.

The complexity lies in finding what technics are the most appropriate for the desire application contrasting between the existing different methods and preparing them for the project Techs4AgeCar of Robesafe group. One of the main specifications of this project is that the principal sensor will be a 3D Lidar, so that the SLAM system can be integrated in the localization module of the autonomous vehicle, which is actually composed by different sensors like odometry, GPS and inertial systems.

Keywords: SLAM, lidar, autonomous navigation, ROS, mapping, localization, CARLA, CARLA-ROS Bridge, Cartographer, Blam, Hdl_graph_slam, Loam.

Resumen extendido

La tecnología aplicada a la conducción autónoma ha ido creciendo exponencialmente en los últimos años. La primera presentación de un vehículo autónomo que se conoce data del año 1939 en la feria de muestras Futurama. A partir de ese momento los sistemas de conducción autónomos se han convertido en complejos sistemas que a través de sensores y cámaras permiten realimentar el sistema de conducción.

La conducción autónoma se basa en una serie de reglas y normas donde no existe cabida para la interpretación ni violación de estas, salvo fallo del sistema. Es decir, la gran utilidad de implementar este tipo de modos en la sociedad es la capacidad de reaccionar ante situaciones de peligro en las que el ser humano, quizás no sería capaz de responder.

Este proyecto nace de la necesidad de mejorar el sistema de localización implementado en el vehículo autónomo del proyecto Techs4AgeCar del grupo de investigación RobeSafe. Este sistema integra a través de un filtro de Kalman las medidas del GPS, la IMU y la odometría del vehículo. Sin embargo, en situaciones en las que el GPS no da una información fiable, el sistema sufre importantes derivas. Por ello se pretende incorporar en el filtro de Kalman las medidas de un nuevo sensor, en este caso un Lidar 3D.

Pero, previamente se debe realizar un estudio de las diferentes técnicas de SLAM empleadas hoy en día para poder decidir cuál es la que mejor se ajusta a la aplicación. A través de artículos científicos y paginas como las de Robot Operating System (ROS) o relacionadas se puede encontrar diferentes técnicas o algoritmos que podrían ser válidos para la aplicación.

Entre todos ellos se han escogido cuatro métodos que se someterán a estudio: *Google Cartographer* (desarrollado por Google), *Loam* (desarrollado por Ji Zhang), *Blam* (desarrollado por Erik Nelson) y *HDL-graph-slam* (desarrollado por Kenji Koide).

El criterio para escoger estos algoritmos es que todos ellos disponen de implementaciones en código abierto dentro del entorno de desarrollo ROS, que es el utilizado dentro del proyecto Techs4AgeCar. Se analizarán las ventajas e inconvenientes que ayudarán a decidir cuál es la técnica escogida para implementar en este proyecto.

Las simulaciones de todas las técnicas se efectuarán utilizando CARLA, un novedoso e hiperrealista simulador de vehículos autónomos que simula una amplia gama de condiciones de conducción y sensores robóticos. Además, a través de su entorno virtual podemos probar diferentes condiciones climáticas y de iluminación.

Para comunicar los datos del simulador CARLA con cada uno de los algoritmos, se empleará Carla-Ros-Bridge, de manera que se podrá generar una nube de puntos con los datos del Lidar 3D simulados por CARLA en el formato requerido por los topics de ROS que posteriormente permitirá obtener el mapa en el que el coche tendrá que localizarse.

Gracias a la herramienta de visualización de ROS en 3D llamada RVIZ, que permite probar el sistema en un entorno de trabajo que se asemeja al mundo real, se puede comprobar el funcionamiento de cada uno de los métodos.

Capítulo 1
Introducción.

La finalidad principal de este trabajo es estudiar y comparar diferentes técnicas de localización y mapeado simultaneo basadas en lidar 3D, con el objetivo de escoger la que más se amolda a las necesidades de proyecto Techs4AgeCar, evitando que el vehículo autónomo sólo obtenga las medidas de posicionamiento a través del GPS y la odometría y consiguiendo de esta manera una localización mucho más robusta y fiable.

1.1. Entorno de trabajo y motivación

En el grupo de investigación Robesafe [1] de la Universidad de Alcalá llevan años incorporando sistemas y metodologías que conformen un vehículo eléctrico autónomo robusto.

En la actualidad el desarrollo del vehículo autónomo del grupo Robesafe se encuentra dentro del proyecto Techs4AgeCar, financiado por el Ministerio de Ciencia, Innovación y Universidades Español, los fondos FEDER y el proyecto RoboCity2030-DIH-CM. El vehículo debe ser capaz de conducir de manera autónoma por entornos reales.

El objetivo final de la construcción de este tipo de vehículo es la obtención de un medio de transporte para personas mayores, que por problemas de movilidad no puedan conducir por sí mismos un coche como medio de transporte.

Este Trabajo Fin de Máster únicamente se centrará en la parte de los sensores del vehículo autónomo, ya que son los elementos que darán la información suficiente para aplicar las técnicas de SLAM al coche.

El vehículo está equipado con un Lidar 3D de tipo Velodyne de 16 [2] haces situado en la parte superior de este y mirando hacia el frente, también dispone de una cámara de stereo visión a color ubicado en el parabrisas delantero, y por último una GPS TopCon HiperPro DGPS-RTK instalada en la parte superior en la trasera central del vehículo.

En capítulos posteriores se procederá a explicar más detalladamente cada uno de los sensores disponibles.

La motivación para la realización de este trabajo viene dada por el problema actual que existe con el sistema de localización del vehículo. Hoy en día el vehículo por medio de un filtro de Kalman integra las medidas del GPS y la odometría. El contratiempo se da cuando el GPS no es capaz de dar una información fiable y el sistema sufre pérdidas de la información. Sí en vez de depender únicamente del sistema GPS incorporamos un sensor láser, el vehículo podrá reconducirse ante posibles derivas y además podrá alinear correctamente su posición con el mapa del entorno disponible previamente.

La siguiente imagen muestra el vehículo autónomo del grupo Robesafe de la Universidad de Alcalá, sobre el que se pretende incluir la técnica de localización y mapeado simultáneo seleccionada en este TFM.

Ilustración 1 Vehículo autónomo grupo Robesafe, Universidad de Alcalá

1.2. Objetivos del trabajo

Una vez descrita la motivación de este trabajo, se detallan a continuación los principales objetivos específicos a abordar.

- Realizar un estudio previo sobre la teoría de los algoritmos de SLAM y el entorno de desarrollo para aplicaciones robóticas Robot Operating System [3].
- Realizar una búsqueda de algoritmos de SLAM basados en Lidar 3D como sensor principal, seleccionado para su estudio aquellos que disponen de implementaciones en código abierto en ROS.
- Estudiar por separado cada uno de los algoritmos seleccionados y comprobar su funcionamiento, utilizando para ello dos entornos de pruebas: (1) el simulador hiperrelástico CARLA y (2) los datos reales obtenidos de los sensores del vehículo Techs4AgeCar.
- Realizar una comparativa de los algoritmos y establecer las métricas necesarias que nos ayuden a seleccionar el método óptimo para integrar dentro del sistema de localización del vehículo Techs4AgeCar.
- Integrar el sistema de SLAM escogido en el filtro del Kalman del vehículo y comprobar las mejoras obtenidas en la localización.

1.3. Estructura del trabajo

Este TFM está dividido en varios capítulos y secciones, de los cuales esta introducción se corresponde con el primero, donde se presenta el entorno de trabajo y los objetivos del proyecto.

En el segundo capítulo se realiza un estudio del estado del arte en el área de los sistemas de SLAM basados en lidar 3D, las implementaciones para ROS, así como una introducción al problema del SLAM.

En el tercer capítulo se desarrollan las herramientas utilizadas, tanto el entorno de desarrollo ROS como el simulador CARLA, además de una descripción de las distintas técnicas de SLAM que van a ser sometidas a estudio.

El cuarto capítulo desarrolla en profundidad cada uno de los métodos bajo estudio, explicando el algoritmo utilizado por cada uno de ellos, así como los resultados en el simulador CARLA.

El capítulo 5 recoge los resultados finales en entorno real utilizando cada uno de los métodos bajo estudio, y una comparativa de los mismos con el Ground Truth de CARLA.

Las conclusiones del proyecto están recogidas en el capítulo 6, añadiendo algunas líneas futuras que pueden derivar de este trabajo.

En el apartado de anexos se encuentran el manual de usuario, el presupuesto y el pliego de condiciones.

Capítulo 2.
Estado del arte

2.1. Introducción

A nadie le sorprende que hoy en día se hable de vehículos autónomos, cada vez más son las noticias que se hacen eco de los avances de este sector automovilístico. Grandes empresas destinan enormes sumas de dinero para mejorar sus modelos y obtener grandes ventas en el futuro.

Como es evidente la seguridad es una parte primordial a la hora de constituir este tipo de vehículos. Por eso es importante aplicar metodologías y modelos capaces de resolver los imprevistos que pueden surgir a la hora de conducir tanto por carretera como por ciudad.

En este capítulo se realiza un estudio del estado de arte dentro del contexto del presente trabajo. Para ello se han revisado una gran cantidad de páginas web, artículos científicos y portales investigadores.

2.2. El problema del SLAM

La comunidad investigadora lleva años intentando resolver el problema que presentan los sistemas de localización y mapeado simultáneos (SLAM) [4], puesto que conforman una parte esencial de los sistemas de conducción autónoma, dotando al vehículo de mapas y localización actualizada al momento.

En SLAM los sensores hardware utilizados para implementarlo se pueden dividir en dos categorías. Los sensores propioceptivos captan información interna

propia del vehículo (velocidad, aceleración, orientación) y los exteroceptivos captan la información del exterior. Ambas informaciones se combinan para localizar y mapear. Tanto los encoders como la IMU son empleados para obtener una estimación a corto plazo de la trayectoria seguida por el vehículo.

Para el proceso de mapeado el sistema necesita detectar la distancia entre el robot y los objetos de su alrededor, a través de láser, cámaras etc...

El problema de localización y mapeado simultáneo plantea la posibilidad de que, al situar el vehículo en una posición desconocida dentro de un entorno también desconocido, este pueda ser capaz de construir un mapa incremental del entorno y a la vez saber cuál es la posición exacta en la que se encuentra dentro de dicho mapa.

En la conducción autónoma la manera que tiene el vehículo de obtener información es a partir de los sensores y del conocimiento del movimiento propio. Es por ello por lo que existen múltiples fuentes de incertidumbre que pueden llevar al vehículo a cometer errores que si sucedieran en el mundo real podrían ser fatales. Algunos de ellos son:

- El ruido de los sensores.
- Simetrías de entorno: El entorno sobre el que se mueve el vehículo puede presentar coincidencias que dificultan la tarea de localización del vehículo.
- Cambios en el entorno: Cuando se producen cambios en los lugares por los que circula el vehículo, estos pueden llevar a un error de posicionamiento, debido a que el mapa podría estar desactualizado.
- Aproximaciones de los modelos: Al emplear modelos probabilísticos es posible que, al realizar aproximaciones, estas provoquen un error acumulado.

La suma de todos estos errores hace que adquirir un sistema de SLAM sea una ardua tarea que requiere algoritmos complejos.

El gran problema al que se enfrentan los sistemas de SLAM es el debido a la correlación existente entre el error de localización y el error del mapa. Es decir, para obtener el mapa es necesario conocer la localización del robot móvil y para saber la posición del robot se requiere de un mapa actualizado.

El cierre de lazos también causa bastantes problemas a la hora de mejorar y poner a punto los sistemas de SLAM. Al fin y al cabo, cada vez que el vehículo circula por una misma sección del entorno y lo repite de manera constante, los mapas que se van guardando deben superponerse para generar un mapa consistente. Es decir, cada vez que se crea un nuevo mapa, la resolución de este ira incrementándose de manera exponencial al poseer más información, tanto de su vuelta actual como de las pasadas. El problema se halla al querer superponer todos los mapas que ha ido recogiendo el vehículo, puesto que todos ellos poseen desplazamientos generados por todas las incertidumbres que se dan por los errores de los sensores ya comentados. Como ejemplo, la siguiente muestra, donde en la imagen de la izquierda la superposición de los mapas generados en cada vuelta realizada por el vehículo, generar un mapa incorrecto por un error de desplazamiento en todos ellos.

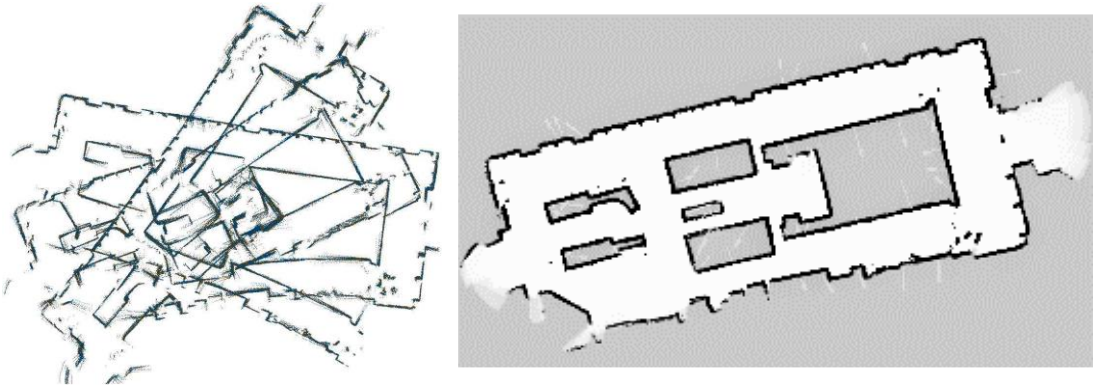

Ilustración 2 Error lazo de control aplicando técnicas de SLAM

Por todo ello no es de extrañar que los métodos que solucionan el problema descrito estén basados en modelos probabilísticos [5].

El problema del SLAM se puede subdividir en diferentes tareas:

1. Obtener la información que aportan los sensores.
2. Detectar cuales son los puntos de interés del entorno a partir de las referencias marcadas.
3. Implantar correspondencias entre lo esperado y lo observado.
4. Calcular la posición.

2.3. Sistemas del SLAM basados en Lidar 3D

Existen multitud [6] de grandes empresas y universidades que implementan las técnicas de SLAM en sus proyectos de investigación para la dotación autónoma de sus vehículos en exteriores. Algunos ejemplos son:

- El proyecto *CityMobil2* recorrió más de 26000 kilómetros durante varios en meses en diferentes ciudades, para poder obtener las suficientes medidas y generar un mapa de gran resolución.

- La Universidad de Stanford participó en el *Urban Challenge* [7]. Su vehículo generaba un mapa terrestre utilizando GPS, IMU y un Lidar 3D de 64 capas de múltiples escaneos sin conexión. La localización se llevó a cabo dentro del mapa con un filtro de histograma 2D con una precisión de 10 cm.

Ilustración 3 Vehículo autónomo de Stanford e histograma 2D en tiempo real

- La Universidad de ULM [8], creaba un mapa compuesto por características MSER procedentes del sistema de visión del coche y de una rejilla Lidar 2D georreferenciada mediante un GPS RTK. La localización se llevó a cabo a través

del método Montecarlo. Las pruebas se realizaron a lo largo de 5 km, todas ellas con una precisión de 10 cm.

- BMW [9], ofrece una descripción general de su experiencia con la conducción autónoma durante miles de kilómetros en la vía pública. Se basan en el método del carril marcado mediante el uso de visión, Lidar, así como odometría y GPS. Para la detección de los límites de la carretera, necesitan Lidar y sensor de radar. Su mapa integra información semántica y geométrica (carril, modelos, conectividad, etc) así como datos de localización (marcas de carril y posición de los limistes de la carretera). En BMW, los mapas a gran escala son una preocupación y por ello deben desglosarse en submapas.

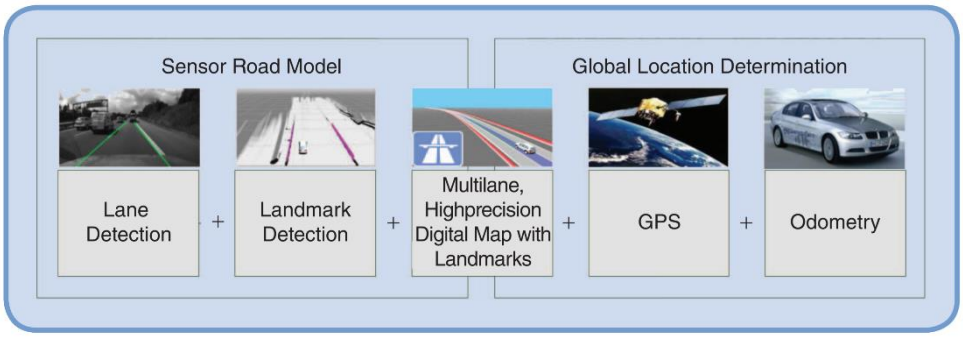

Ilustración 4 Fusión sensorial aplicada en BMW [9] para localización.

2.4. Implementaciones para ROS

A lo largo de las últimas décadas, son múltiples los hallazgos realizados en este campo de la robótica.

Para seleccionar el algoritmo que más se ajuste a la aplicación para la que está destinada este TFM, se debe realizar un estudio de las diversas técnicas disponibles para localización y mapeado simultaneo de conducción autónoma.

En este caso, el vehículo autónomo del que dispone el grupo Robesafe, dispone de un láser 3D de 16 haces (modelo Velodyne), por lo que los sistemas de SLAM entre los que debemos seleccionar los mejores, deben tener la condición indispensable de utilizar como sensor principal un láser de esas características.

2.4.1. Mrpt Localization

Mobile Robot Programming Toolkit [10] [11] [12], es un algoritmo formado por un conjunto de paquetes y aplicaciones software para robótica móvil. Esta técnica tiene en cuenta la forma 3D del robot y además ofrece la posibilidad de realizar trayectorias no circulares, lo que conforma un gran abanico de posibles movimientos para lograr alcanzar el destino deseado.

Emplea un filtro de partículas para un robot móvil usando la odometría y un conjunto de balizas en posiciones conocidas. Este filtro permite la localización con:

- Diferentes algoritmos.
- Diferentes formatos de mapas: el filtro dispone de diversas entradas tales como nube de puntos, mapa de baliza o mediante rejillas de ocupación.
    - Rejillas de ocupación: para generar este tipo de mapa se utilizan sensores que sean capaces de que, a través de la generación de una nube de puntos, se estimen varias rejillas a diferentes alturas.
    - Mapa originado con balizas en posiciones 3D predefinidas. Únicamente emplea sensores de rango.
- Múltiples sensores simultáneos: su combinación se puede utilizar a la vez junto con la información probabilística.

MRPT adopta únicamente como objetivo la posición de la meta y no la orientación, al contrario de la mayoría de los planificadores de ROS.

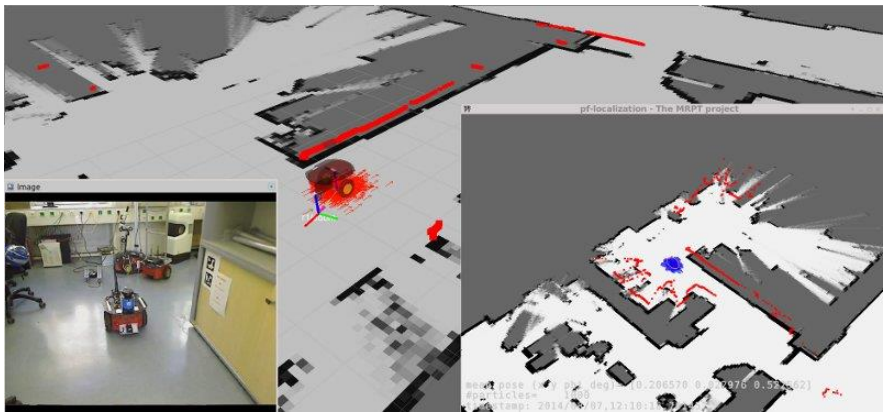

Ilustración 5 Mrpt Localization

2.4.1.1. *Desventajas*

En lo que concierne a este proyecto TFM, la mayor desventaja o inconveniente de este método es el hecho de no poder funcionar en tiempo real.

Puesto que la aplicación final será incorporar la técnica seleccionada en el vehículo autónomo del grupo Robesafe, este requiere un funcionamiento tanto en entorno simulado como en entorno real. El coche necesita un mapa y una posición actualizada en todo momento.

Otra razón que ha provocado que definitivamente se decida descartar este método para llevarlo a cabo en el grupo de investigación, es el hecho de que al obtener un mapa 2D y localización 2D, se pueda producir indeterminismo, pues no se contemplarían rampas, cuestas, dobles alturas etc, que al fin y al cabo son elementos del entorno real muy habituales.

2.4.2. Gmapping

Gmapping [13] [14] es el algoritmo de SLAM más integrado que se pueda usar en ROS hoy en día. Para ello emplea un nodo denominado *slam_gmapping* que permite crear un mapa de ocupación 2D en formato rejilla, gracias a la información del láser y la posición acumulada del robot móvil.

Los algoritmos de SLAM suelen incorporar filtros de optimización y el caso de Gmapping no es una excepción. Este método utiliza el filtro de partículas Rao-Blackwellized [15].
Rao-Blackwellized asume que, dada la trayectoria del observador, cada partícula puede llevar un mapa individual del entorno. El número de partículas se ve reducida computacionalmente como resultado de una distribución probabilística que tiene en cuenta el movimiento del robot y las observaciones más recientes proporcionadas por los sensores. Todo ello lleva a que la incertidumbre sobre la posición del robot se reduzca drásticamente.

Gmapping utiliza el algoritmo *Split-and-merge* , una técnica de procesamiento de imágenes. El procedimiento se puede ver en la imagen mostrada a continuación.

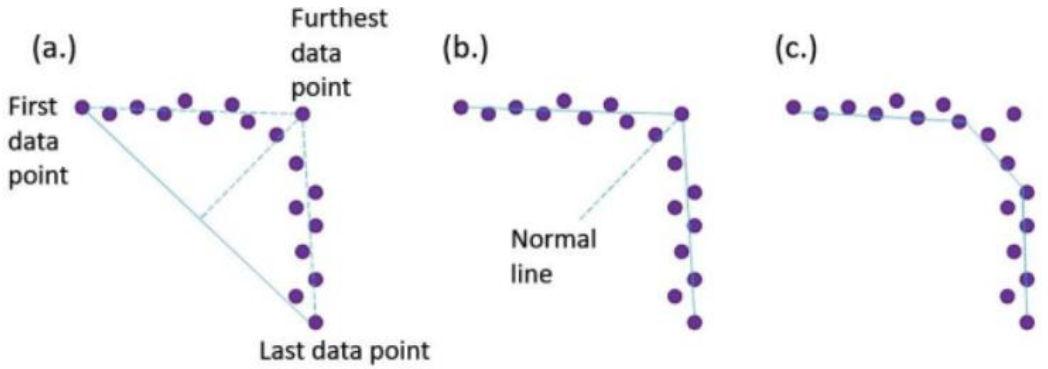

Ilustración 6 Split-and-merge algorithm. Gmapping

En primer lugar, se establece una línea de estimación que conecta el punto inicial con el punto final. A continuación, se une el punto más lejano con la línea que une punto inicial y final, si la longitud de la línea es mayor que un umbral predefinido, la línea se divide en dos, conectando el primer y último punto con el punto más lejano. Este procedimiento se repite hasta que la longitud de la línea normal es menor que en umbral dado.

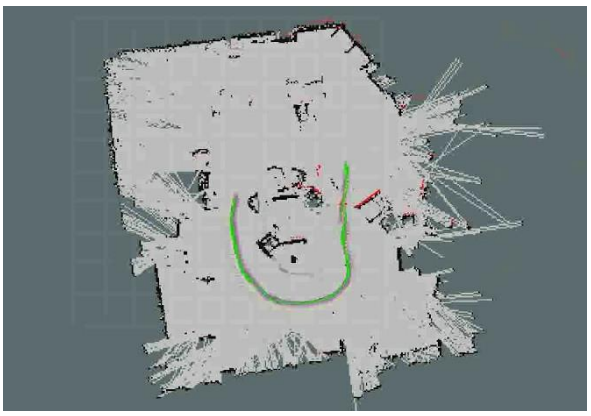

Ilustración 7 Gmapping SLAM

2.4.2.1. Desventajas

Al igual que en el algoritmo MRPT Localizatión, el hecho de que solo se puedan generar un mapa 2D supondría un problema para poder implementarlo posteriormente en el vehículo autónomo, que pretende aprovechar la información del Lidar 3D para captar más información sobre un entorno complejo.

2.4.3. Hector Slam

El algoritmo Hector Simultaneous Localization and Mapping [16] [17] es un sistema de SLAM 3D, que combina el escaneo de lidar 2D con información de una IMU mediante un filtro de Kalman (EFK).

Está compuesto por tres módulos diferentes:

- hector_mapping: el nodo de SLAM.
- hector_geotiff: donde se guarda el mapa y la trayectoria del robot.
- hector_trajectory_server: guarda las transformadas de Fourier (TF) de las trayectorias.

El sistema no está diseñado para requerir información sobre la odometría, por lo que únicamente confía en los datos proporcionados por el sensor láser y una IMU. Combinando la posición con la unidad pitch/roll, el láser consigue estabilizarse provocando que el sistema sea capaz de trabajar en entornos no necesariamente planos.

Uno de los nodos del paquete fusiona el Filtro Extendido de Kalman (EKF) con la información proveniente de la IMU y del error de posición 2D del láser. El filtro se basa en un modelo genérico del movimiento para vehículos terrestres y es mejorado gracias a la IMU, sin tener que depender de los datos de la odometría que normalmente pueden ser erróneos debido a las irregularidades del terreno, o inexistentes como en el caso de vehículos aéreos.

Con la nube de puntos formada por el láser, se genera un mapa de elevación de tipo cuadricula 2D que almacena el valor de la altura correspondiente a cada celda.

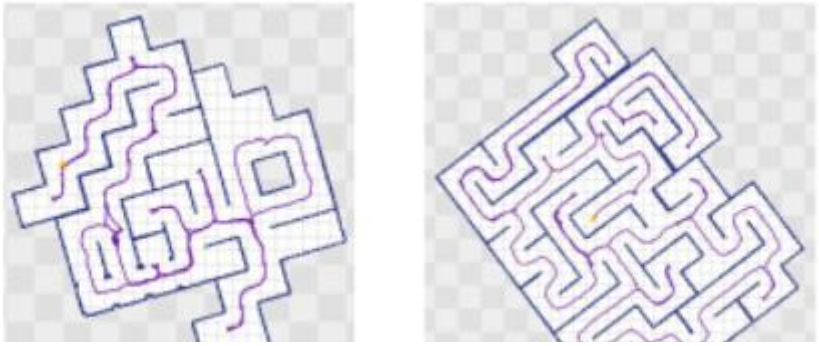

Ilustración 8 Mapa generado por Hector SLAM

Finalmente, otro de los nodos fusiona el mapa de elevación 2.5 D con el mapa de ocupación de rejilla 2D.

Las posiciones por las que ha pasado el robot se guardan en uno de los nodos y se muestrean en función de la distancia entre si agregándolos a una lista. Un algoritmo de transformación aplicado a dicha lista, junto con una búsqueda de la celda con el valor más alto, consiguen generar un punto lo suficientemente alejado y seguro para alcanzar el robot.

2.4.3.1. Desventajas

Hector SLAM es un paquete muy completo que ofrece múltiples opciones de trabajo, sin embargo, no se ha decido llevar a la fase de implementación. Este método, aunque en este caso sí que obtiene un mapa 3D, lo hace utilizando un lidar 2D combinado con una IMU. Para este proyecto se busca un algoritmo que utilice lidar 3D, que contiene mucha más información del entorno.

2.4.4. MCL_3dl

MCL_3dl [18] es un nodo de ROS que incorpora un sistema probabilístico con 6 grados de libertad de la localización de un robot móvil con un sensor láser 3D. Como requerimiento previo necesita una nube de puntos, la cual está basada en el algoritmo Monte Carlo Localization (MCL). La nube de puntos se pude obtener a partir de otros algoritmos que se dedican únicamente a la parte de mapeado del entorno.

MCL o filtro de partículas de localización representa una distribución probabilística de la posición del robot dotando de pesos y densidades a las partículas y con ello estimando el punto exacto de interés.

La nube de puntos de referencia es recibida por el algoritmo como un mapa del entorno y este se localiza a través de los 6 grados de libertad de las posiciones medidas con la ayuda de una predicción del movimiento gracias a la odometría.

Existen dos modelos que se encargan de reducir los falsos positivos de coincidencias en el mapa entre los puntos medidos y observados, *beam_range_finder* y *likelihood_field_range_finder_model*, siendo el primero de los dos más pesado computacionalmente. MCL_3dl utiliza ambos modelos.

En esta implementación la probabilidad de *likelihood_*field_range_finder_model es la suma de las distancias desde cada punto medido al punto más cercano en el mapa. La probabilidad de *beam_range_finder* se calcula mediante el algoritmo Ray Casting.

La predicción de la posición de las partículas se estima gracias a la información de los datos de odometría con parámetros de ruido. Como consejo se recomienda utilizar los datos de odometría compensados por la IMU.

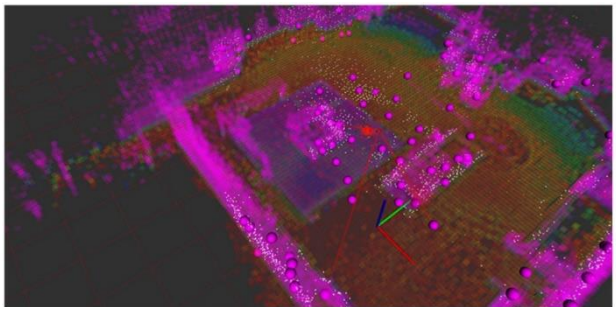

Ilustración 9 MCL_3dl

2.4.4.1. Desventajas

Este algoritmo no construye su propio mapa, si no que requiere de uno previamente construido por otro método, realizando localización sobre él. Por lo tanto, no es un algoritmo de SLAM completo.

2.5.5. Google Cartographer

Google Cartographer [19] es un paquete de Google de código abierto. Es un sistema que permite realizar localización y mapeado simultáneo en tiempo real en entornos tanto de exterior como de interior. Cartographer ofrece la posibilidad de obtener dos tipos de mapa (2D o 3D) según el sensor láser utilizado.

Sus creadores publicaron los datos de dos sensores láser y de un sensor IMU, recopilados durante más de tres años. Las pruebas se realización en el *Deutsches*

Museum, por lo que uno de los primeros pasos a realizar antes de implementar los datos propios, es probar el algoritmo con el ejemplo generado por ellos.

Cartographer divide el problema del SLAM en dos partes igual de importantes: Local SLAM y Global SLAM. En posteriores capítulos se explicarán más detalladamente las dos partes que componen a Google Cartographer.

Los resultados pueden verse a través de la herramienta Rviz, la cual permite visualizar los diferentes submapas que luego conformarán el mapa del entorno.

Son requisitos imprescindibles, poder disponer de datos proporcionados por un láser 3D, información de un sensor IMU y datos de odometría.

Este método ha sido seleccionado para el estudio en este TFM.

2.5.6. Loam SLAM

Loam es un método de SLAM que permite mapear en tiempo real, obteniendo la odometría necesaria para saber la ubicación del vehículo/ robot con el que se esté trabajando.

Al igual que Cartographer, Loam divide el problema del SLAM en dos partes, una se encarga de calcular la odometría y otro de generar buenas coincidencias entre las distintas nubes de puntos para crear un mapa del entorno lo suficientemente realista.

Es condición indispensable disponer de un láser 3D para la creación de nube de puntos del entorno que rodea al vehículo.

Todos los parámetros del algoritmo se pueden a justar según las condiciones que tenga el usuario para hacer funcionar su aplicación final.

Este método ha sido seleccionado para el estudio en este TFM.

2.5.7. Hdl_Graph_SLAM

Hdl_Graph_SLAM es un paquete de ROS de código abierto en tiempo real que intenta resolver el problema del SLAM utilizando para ello un láser 3D. Se basa en un método muy conocido como es Graph_SLAM 3D, disponiendo de un algoritmo de coincidencia de los escaneos llamado NDT.

Se puede generar mapas tanto de entornos de exterior como de interior, con gran resolución de los elementos que lo rodean.

Existen varias restricciones, que pueden ser activadas o desactivadas cambiando los parámetros del archivo launch de este algoritmo:

- Odometría.
- Cierre de lazo.
- GPS.
- IMU.
- Plano del suelo.

Además, existen paquetes relacionados como son Hdl_localization y Hdl_graph_trackig, también a disposición del usuario y de código abierto.

Este método ha sido seleccionado para el estudio en este TFM.

Capítulo 3.
Herramientas utilizadas.

A lo largo de este capítulo se van a explicar todas las herramientas utilizadas para llevar a cabo la implementación de las técnicas de SLAM seleccionadas.

3.1. Simulador CARLA

Como ya se ha comentado en capítulos anteriores para realizar las pruebas con los algoritmos seleccionados en entornos simulados, se trabajará con el simulador CARLA [20].

Entrenar o investigar sobre vehículos autónomos en entornos reales supone un alto coste de infraestructura y dificultades logísticas que no todos los grupos de investigación podrían permitirse. Igualmente, un solo vehículo no sería suficiente para recoger todos los datos necesarios y para cubrir la multitud de casos posibles que se pueden dar en la conducción. Por eso, como alternativa se ofrece la posibilidad de trabajar en simulación y preparar los coches para el momento en el que tengan que ponerse en marcha en la vida real.

CARLA se creó para ser útil como fuente de entrenamiento, validación o desarrollo de proyectos de conducción autónoma en entornos urbanos. Al ser gratuito y de código abierto supone una manera de fomentar la investigación y desarrollo de este sector.

Carla es un simulador de código abierto creado para la investigación de la conducción autónoma. Posee diferentes escenarios con un grado de complejidad que puede ir incrementándose, y con condiciones climáticas y de iluminación desde lluvia, nubes, hasta atardeceres o días soleados como se puede ver en la Ilustración 10. El usuario puede escoger tanto la cantidad de tráfico, el clima, el comportamiento de los peatones, qué sensores utilizar, etc.

Ilustración 10 Mapas urbanos ante cambios de las condiciones climáticas. CARLA Simulator

Además, a pesar de ser una simulación, el entorno se acerca bastante a lo que es la realidad, pues se pueden ver desde coches y edificios, hasta elementos urbanos como farolas o papeleras, pasando por vegetación o grandes infraestructuras. Incluye 40 edificios diferentes, 16 modelos de vehículos animados y 50 peatones animados. El hecho de conservar la escala original provoca una sensación de mayor hiperrealismo (Ilustración 11).

En CARLA, los peatones no siguen pautas fijas, sino que se mueven por el entorno de forma aleatoria de un punto a otro de las calles y por los pasos de peatones. Cada vehículo está pintado al azar para evitar el uso de patrones.

Ilustración 11 Mundo CARLA simulator

Al final existen momentos puntuales que pueden suponer riesgo de accidente a los que un conductor tiene que enfrentarse (niños cruzándose en medio de la calle, vehículos que frenan inesperadamente, otros conductores que no respetan las reglas de conducción, un vehículo en sentido contrario, etc). Los creadores de CARLA son conscientes de todas ellas y con su simulador se repiten esas situaciones de peligro de manera interminable para ayudar al aprendizaje del vehículo.

CARLA simulator estudia tres enfoques distintos de conducción autónoma. El primero de ellos es un pipeline modular clásico, el segundo es una red neuronal que aprende por imitación y, en tercer lugar, también una red neuronal entrenada de un extremo a otro a través de aprendizaje reforzado.

El simulador CARLA admite una configuración flexible de sensores y señales utilizadas para entrenar estrategias de conducción, tales como velocidad, aceleración o coordenadas GPS.

CARLA dispone de múltiples sensores, pero los más comunes son: cámaras RGB (pseudo-sensores que permiten segmentación semántica, o *ground-truth*, Lidar y GPS.

Entre los parámetros que incluyen las cámaras se encuentran: posición y orientación 3D con respecto al coche, campo de visión y profundidad del entorno.

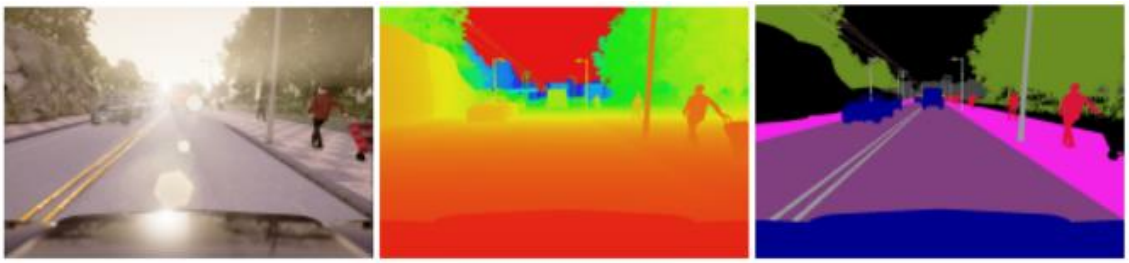

Ilustración 12 Modalidades de sensores CARLA simulator

Además de la lectura de los sensores y pseudo-sensores, CARLA proporciona un amplio abanico de medidas asociadas con el estado del agente, como por ejemplo la ubicación, la orientación del vehículo con respecto al sistema de coordenadas del mundo, velocidad y una acumulación de las colisiones. También incluye el estado de los semáforos, el límite de velocidad etc.

En el caso de este TFM, únicamente será necesario utilizar el Lidar 3D que nos proporciona la nube de puntos, la Unidad de Medida Inercial (IMU), ya que alguna de las aplicaciones estudiadas tiene como requisito su utilización, y lo mismo sucede con la odometría.

Una de las ventajas de utilizar CARLA es que es capaz de integrar ROS a través de *CARLA ROS Bridge*.

3.2. Robot Operating System

Robot Operating System (ROS) [3] es una plataforma de desarrollo robótico que permite crear aplicaciones con múltiples sensores y actuadores de forma flexible [21]. Es un conjunto de herramientas, bibliotecas y convenciones que tienen como finalidad simplificar la tarea de crear un comportamiento de robot complejo. Permite programar tanto en C como en C++ y otros lenguajes de programación, destacando Python.

Robot Operating System se creó desde cero a través de diferentes grupos de investigación que se dedicaban a desarrollar partes del software de robótica que después se unían para formar lo que hoy en día es la plataforma ROS.

En la Ilustración 13 se ve el esquema general de programación empleando plataformas robóticas. Como se puede observar la plataforma robótica se sitúa en un nivel intermedio entre las aplicaciones de usuario y la plataforma física (o simulador), simplificando el acceso como ROS incorporan un completo y creciente repositorio de algoritmos robóticos aportados por la comunidad científica, que simplifican enormemente la creación de aplicaciones robóticas complejas.

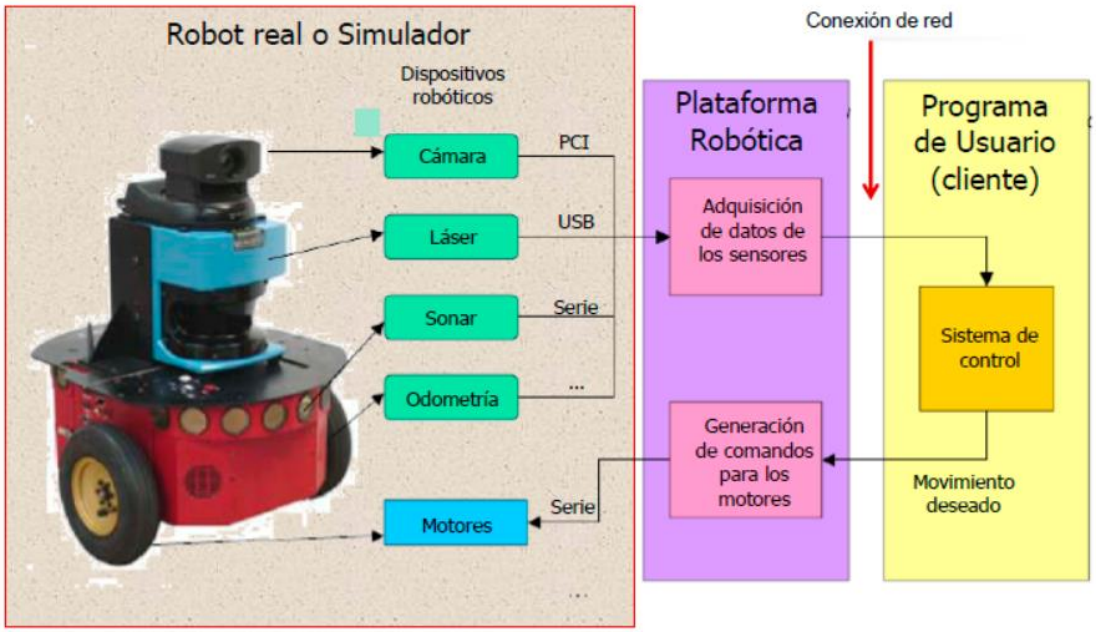

Ilustración 13 Programación empleando plataformas robóticas

ROS es una plataforma de desarrollo que se organiza en distribuciones anuales de forma similar a la distribución de Linux Ubuntu. Las distribuciones liberadas en años pares son LTS (para cada LTS de ROS hay una versión ligada de LTS de Ubuntu), y las liberadas en años impares tienen un soporte de dos años.

En este Trabajo Fin de Máster se va a utilizar la versión LTS ROS Kinetic Kame (2016) funcionando bajo Ubuntu 16.04 LTS.
ROS está basado en una arquitectura de grafos donde el procesamiento de la información pasa a través de nodos que pueden percibir y mandar mensajes de sensores, control, estados, actuadores, entre otros [3]. ROS es capaz de dotar de los servicios estándar de los sistemas operativos como control de dispositivos de bajo nivel, paso de mensaje entre procesos, etc.

3.2.1. Conceptos básicos de ROS

A continuación, se enumeran los conceptos más básicos de ROS:

- ROS Master:

  Se considera el gestor de ROS (Ilustración 14), ya que permite que los nodos se comuniquen entre sí, registra los nombres de los topics, servicios etc. Debe estar siempre presente en el sistema y para ejecutarlo se utiliza la orden *roscore*.

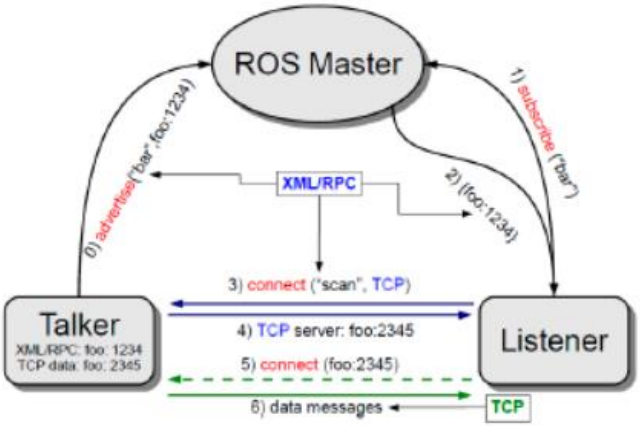

Ilustración 14 Comunicación ROS Master

- Nodos:

  Son fragmentos del código que se compilan, ejecutan y manejan de manera independiente. Pueden suscribirse o publicarse en los *topics* y pueden proporcionar o utilizar servicios.

- Topics:

  Son los canales de información utilizados por los nodos. Cada *topic* puede transportar un único tipo de mensajes.

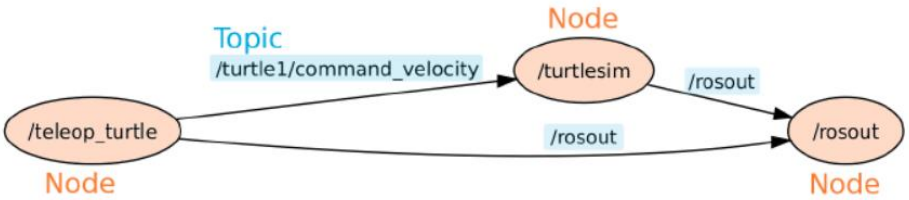

Ilustración 15 Esquema Topics ROS

- Mensajes:

  Son tipos estructurados de datos para la comunicación entre nodos. Cada *topic* sólo puede trasmitir un tipo de mensaje. Existen mensajes ya definidos para los tipos de datos más utilizados (láser, odometría, mapas de ocupación, etc.). Además, pueden definirse nuevos mensajes propios de cada aplicación.

- Servicios:

  Es un modelo de comunicación síncrona entre nodos de tipo cliente/servidor. Se utilizan para eventos especiales o que necesiten confirmación.

- Rosbag:

  Permite almacenar y reproducir datos de una ejecución del sistema. En el caso de este TFM, *rosbag* guardará las grabaciones tomadas en el entorno de simulación de CARLA con toda la información de los sensores necesarios.

- TF:

  Es un mensaje especial de ROS que sirve para relacionar los distintos marcos de coordenadas.

3.2.2. Visualizador Rviz

Rviz es una aplicación de ROS que sirve para visualizar los datos que existen dentro de una ejecución de ROS. Existen diferentes herramientas dentro de RViz que permiten representar datos como odometría, láser, mapas, etc.

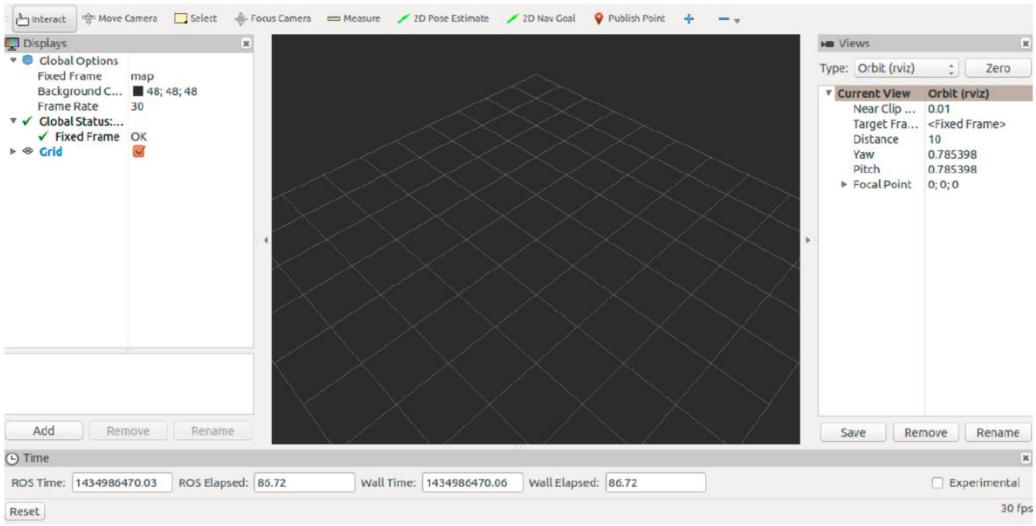

Ilustración 16 Visualizador RViz

En el panel de control se pueden añadir o eliminar *topics*, modificar el marco de coordenadas, etc.

Algunos de los tipos de datos que se pueden visualizar son:

- Mapa:

  Es un mapa 2D de celdas de ocupación y se puede seleccionar su formato (celdas libres, ocupadas y desconocidas).

- Odometría:

  Permite visualizar en 3D la posición actual del robot. RViz permite ver el recorrido de las últimas posiciones.

- Sensor láser:

  Rviz visualiza todos los impactos del barrido del láser y permite colorear por distintos métodos (ejes, intensidades, colores, etc).

- Sensor de distancia:

  Representa los ultrasonidos e infrarrojos y visualiza un cono 3D de la apertura del sensor y también con el tamaño de la distancia medida.

- Árbol de transformadas:

  Representa el eje de coordenadas de cada marco de coordenadas del sistema. Se puede activar o desactivar su visualización en el panel de control.

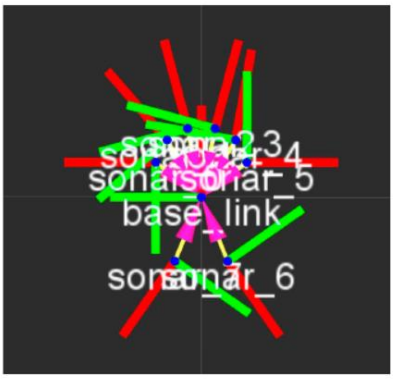

Ilustración 17 Árbol de transformadas RViz

3.3. Paquetes de ROS

En este apartado se explicará más detalladamente cada uno de los paquetes de ROS que se han usado para poner en marcha los algoritmos seleccionados en este TFM. Posteriormente se dedicarán capítulos más extensos concretando las características de los algoritmos.

3.3.1. CARLA-ROS-bridge

CARLA-ROS-bridge [22] es un paquete de ROS que permite establecer dos vías de comunicación entre la plataforma ROS y el simulador CARLA. La información proveniente de CARLA se transfiere a ROS en forma de *topics*, del mismo modo, los nodos de ROS pueden ser traducidos a comandos y ser utilizados en CARLA.

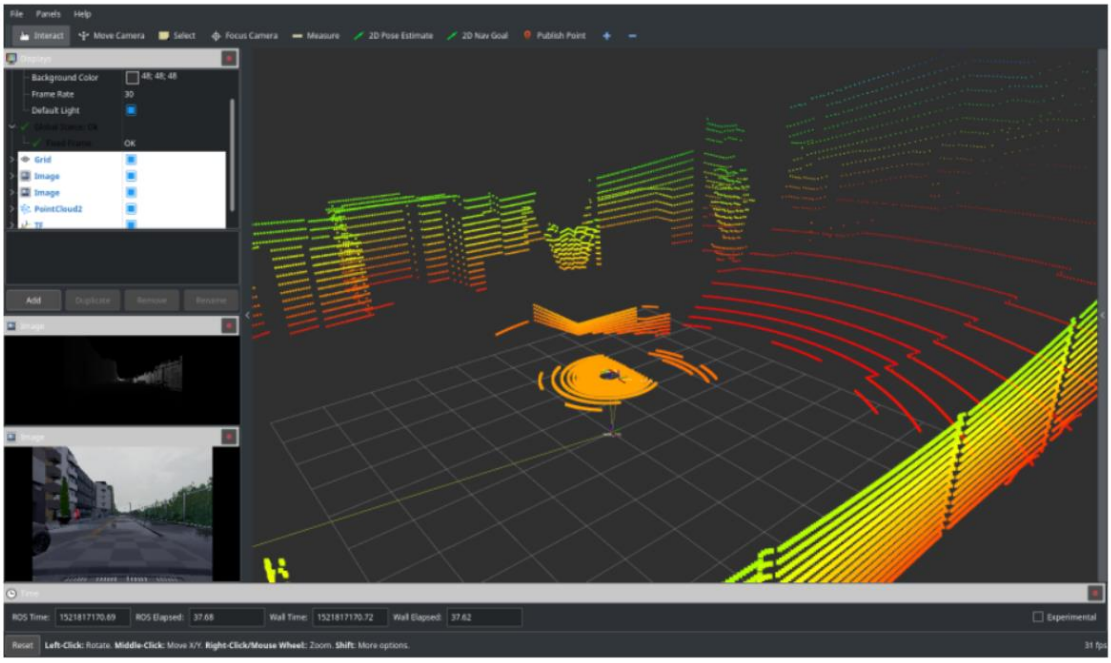

Ilustración 18 Visualizador CARLA ROS-bridge

3.3.1.1. Requisitos

Para poder utilizar este paquete, es necesario tener instalado tanto ROS (Kinetic o Melodic) como CARLA con el número de versión 0.9.7 o versiones posteriores.

3.3.1.2. Características

CARLA-ROS-bridge permite disponer al usuario de:

- Sensores: lidar, cámaras (profundidad, segmentación, RGB), GNSS, radar e IMU.
- Control de los Agentes de conducción autónoma: acelerador, freno y dirección.
- Control sobre CARLA: posibilidad de comenzar o parar la simulación y cambiar los parámetros de simulación.
- Información sobre los objetos: luces de semáforos, visualización de balizas, transformadas, etc.

3.3.2. Cartographer ROS

Google Cartographer es un sistema desarrollado por Google que permite realizar localización y mapeado simultáneo en tiempo real tanto en 2D como en 3D. Dispone de múltiples plataformas y configuraciones de sensores [19] .

3.3.2.1. Introducción

En octubre de 2016, Google lanzó el código fuente de la librería SLAM en tiempo real denominada Cartographer. Los algoritmos utilizados para resolver SLAM en 2D quedaron reflejados en el paper [23]. Los resultados que pudieron ofrecer con este algoritmo fueron impresionantes, especialmente considerando la capacidad de ejecutar en tiempo real con en un hardware básico un algoritmo tan complejo.

Google publicó un banco de datos de alta calidad para que los usuarios pudieran utilizar sus algoritmos y ponerlos a prueba. Gracias a SU integración en ROS, Cartographer pudo y puede ser utilizado en varias plataformas robóticas tan relevantes como Toyota HSR, TurtleBots, PR2 o Revo LDS, permitiendo conectar paquetes de software heterogéneos a través de una comunicación entre procesos estandarizada.

Para ayudar a los usuarios y que estos pudieran disponer de datos y ejemplos reales de su algoritmos, los creadores de Cartographer publicaron los datos de dos sensores láser (VLP-16) y de una unidad de medida inercial (IMU) recopilados durante tres años a través de unas plataformas robóticas que implementaban su mapeado 2D y 3D. Estas pruebas se llevaron a cabo en el *Deutsches Museum* en Munich, Alemania. La trayectoria procesada fue de 108 metros de largo y contenía 300.000 escaneos 3D de los sensores PUCK. La Ilustración 19 muestra el resultado final del mapa aplicando Google Cartographer.

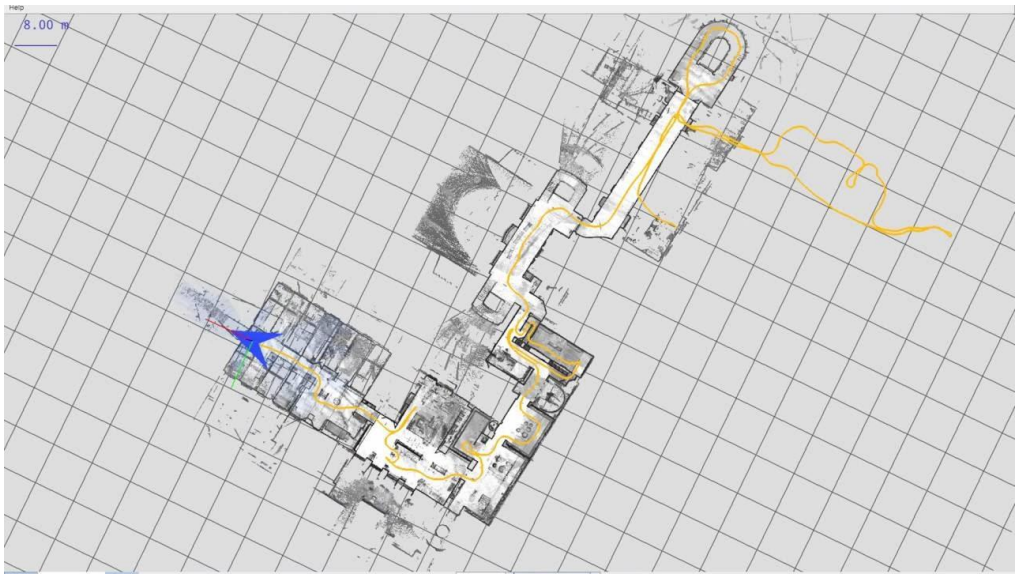

Ilustración 19 Deutsches Musesum Google Cartographer

3.3.2.2. SLAM 2D

Como ya se ha comentado en la sección anterior, Google Cartographer ofrece una solución en tiempo real para mapear estancias en interiores o exteriores a través de una plataforma que genera mapas 2D con una resolución de r=5 cm. El usuario puede observar al mismo tiempo la creación del mapa y la generación de los edificios del entorno que lo rodea.

La forma de trabajar de este algoritmo es a través de los escáneres del sensor láser que se van insertando en submapas (que se supone que son los bastantes precisos durante periodos de tiempo cortos) en la mejor posición estimada.

A continuación, se establece una comparación entre la última información obtenida por el sensor láser y el mapa ya creado gracias a los datos de los escaneos láser pasados, de forma que el algoritmo pueda saber si los trozos nuevos de mapa cuadran en posición con el mapa que se ha ido generado hasta el momento. El error de posición estimada se acumula en el sistema de referencia.

A pesar de ser lo habitual en algoritmos de SLAM, Cartographer no emplea un filtro de partículas para hacer frente al error acumulado, sino que regularmente ejecuta una optimización de la posición.

Cuando un submapa está terminado, los nuevos escaneos del sensor láser no son insertados en él. Este participará en la asociación de escaneos (*scan matching*) para el cierre de lazos, al igual que los escaneos del láser (*laser scan*), de manera sistemática.

Si la plataforma robótica decide volver a pasar por una zona en la que un submapa ya está terminado y los nuevos escaneos del láser están lo suficientemente cerca según la posición estimada actual, una asociación de escaneo (*scan matcher*) intenta encontrar el escaneo en el submapa ya creado. Si esa asociación es una coincidencia lo suficientemente buena, esta es añadida como una restricción del cierre del lazo, utilizada para el problema de la optimización de la posición. Estas coincidencias y sus posteriores asociaciones tienen que ocurrir rápidamente para poder cerrar los lazos de forma suave.

Cartographer ROS proporciona un complemento a Rviz para poder visualizar los submapas. Se puede seleccionar el submapa que se desea ver por su número.

En 3D, Rviz solo muestra proyecciones 2D de las cuadriculas de probabilidad 3D. Se puede elegir entre alta y baja resolución a la hora de visualizar estas cuadriculas.

3.3.3. Loam: Lidar Odometry and Mapping in Real-time

Loam [24] es un método de SLAM en tiempo real para mapear y obtener la odometría utilizando un rango de medidas a partir de un lidar en 2 ejes con 6 grados de libertad.

Loam fue desarrollado en el instituto de robótica en la Universidad Carnegie Mellon por Ji Zhang y Sanjiv Singh en 2014.

Los creadores de Loam consideraron que una buena forma de generar los mapas fuera a través de odometría *lowdrift,* utilizando un lidar con 6 grados de libertad de movimiento. Para ellos la ventaja de utilizar un sensor láser reside en su insensibilidad a la iluminación ambiental y su textura óptica en la escena.

3.3.3.1. Introducción

LOAM logra eliminar los problemas que a menudo suceden con otros métodos de SLAM al trabajar en tiempo real (los sensores transmiten la información de manera descoordinada) sin necesidad de medidas inerciales o rangos de alta precisión. La idea es la división del problema del SLAM en dos algoritmos diferentes. Uno de ellos se encarga de realizar la odometría a alta frecuencia para estimar la velocidad del lidar y corregir la distorsión en la nube de puntos (parte izquierda Ilustración 20). El otro algoritmo trabaja a una frecuencia de magnitud baja para una buena coincidencia y un excelente registro de la nube de puntos, generando el mapa deseado.

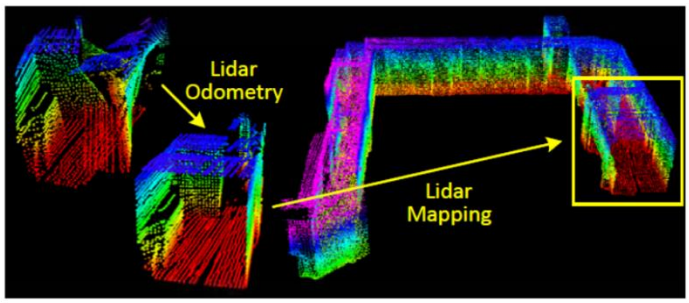

Ilustración 20 Algoritmo LOAM

Ambos algoritmos extraen puntos característicos ubicados en superficies planas y bordes, y hacen coincidir los puntos para generar una nube de puntos de coincidencias.

El algoritmo se divide en dos partes, llevando a cabo la resolución de ambos de manera paralela. El mapeo se lleva a cabo a través de la optimización por lotes (similar al método ICP [25]), para producir una estimación del movimiento precisa y un mapa fiel a la realidad.

Una técnica similar al algoritmo ICP se utiliza para eliminar la distorsión introducida por el láser 3D. Si el movimiento de exploración es lento entonces la distorsión del movimiento puede agravarse.

Al implementar varios sensores es necesario utilizar un filtro de Kalman extendido o un filtro de partículas.

Zhang y Singh asumieron que el láser estaría precalibrado y que las velocidades lineales y el ángulo del lidar serían suaves y continuas a lo largo del tiempo sin cambios bruscos.

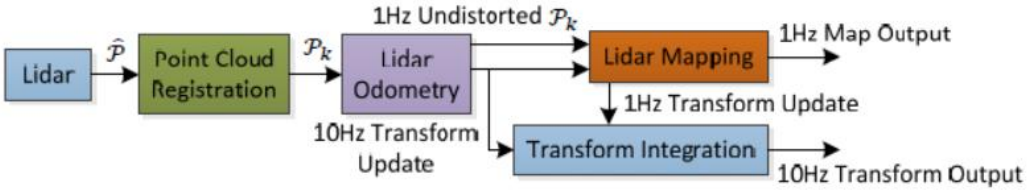

Ilustración 21 Diagrama de bloques del sistema software LOAM

3.3.4. Hdl_graph_slam

Hdl_graph_slam [26] [27] es un paquete abierto de ROS para tiempo real con seis grados de libertad usando un láser 3D. Parte del método Graph Slam en 3D con estimación de odometría y detección de lazos basado en coincidencias. Admite varias restricciones como GPS, IMU (tanto orientación como aceleración), y plano del suelo (detectado gracias a la nube de puntos generado por el láser).

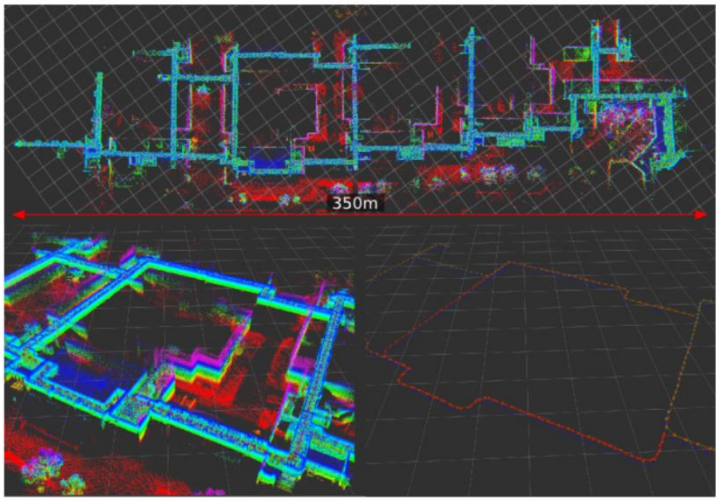

Ilustración 22 Algoritmo Hdl_Graph_SLAM

3.3.4.1. Introducción

Graph SLAM es uno de los algoritmos que más éxito ha tenido a la hora de resolver el problema del SLAM.

El problema de SLAM es resuelto mediante la construcción y optimización de un gráfico cuyos nodos representan parámetros que necesitan ser optimizados (posiciones del sensor, bordes que representan restricciones o posiciones relativas entre la posición del sensor y los puntos de referencia). El gráfico está optimizado para que los errores entre las restricciones y los parámetros se minimicen.

Siguiendo la ecuación que se muestra a continuación:

$$F(x) = \sum \boldsymbol{e}_k(\boldsymbol{x}_k, \boldsymbol{z}_k)^T \Omega_k \boldsymbol{e}_k(\boldsymbol{x}_k, \boldsymbol{z}_k)$$

Donde x_k es el parámetro y z_k es la restricción, siendo $e_k(x_k , z_k)$ el error entre ambos. La ecuación pasa por el algoritmo de Levenberg-Manquardt para su posterior linealización.

El sistema de Graph SLAM, primero estima la trayectoria del sensor aplicando NDT (*Normal Distributions Transform*) a través de la coincidencia de escaneos entre fotogramas consecutivos. Para láser 3D, NDT es uno de los mejores buscadores de coincidencias en términos de velocidad de procesamiento y confianza.

Antes de que se explique con mayor profundidad el algoritmo de Hdl_Graph_SLAM, la Ilustración 23 ofrece una visión global de lo que sería su funcionamiento.

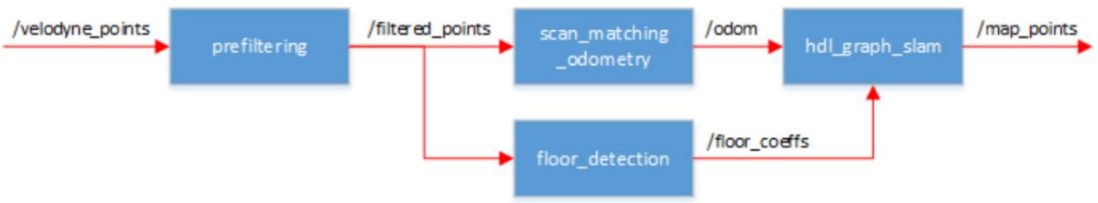

Ilustración 23 Esquema global funcionamiento Hdl_Graph_SLAM

Hdl_Graph_SLAM está compuesto por 4 nodos diferentes:

- prefiltering_nodelet
- scan_matching_odometry_nodelet
- floor_detection_nodelet
- hdl_graph_slam_nodele

En primer lugar, la nube de puntos de entrada se reduce mediante prefiltering_nodelet para pasar a los siguientes nodos. Mientras scan_matching_odometry_nodelet estima la posición del sensor aplicando la coincidencia de escaneos (estimación de la odometría), floor_detection_nodelet detecta los planos del suelo a través de RANSAC [28]. La odometría estimada y los planos del suelo se envían a Hdl_Graph_SLAM.

Para comenzar el error acumulado por la coincidencia de escaneos en la exploración, Hdl_Graph_SLAM optimiza el grafico de posición que tiene en cuenta varias restricciones.

3.3.5. Vehículo Teach4AgeCar

El grupo Robesafe de la Universidad de Alcalá [29] lleva trabajando desde hace años en la construcción de un vehículo eléctrico autónomo. El vehículo se encuentra dentro de los diferentes proyectos en los que el grupo de investigación ha ido participando a lo largo de estos últimos años.

Ilustración 24 Vehículo autónomo grupo Robesafe Universidad de Alcalá

Como se muestra en la Ilustración 25, existen cuatro módulos diferentes dentro del espacio de trabajo software del coche que se comunican a través del sistema de comunicación de procesos internos de ROS. Cada módulo es independiente y procesa la información de manera asíncrona. Los módulos software son:

- La capa del controlador del hardware.
- La capa de control.
- La capa ejecutiva
- La capa de interfaz con el usuario.

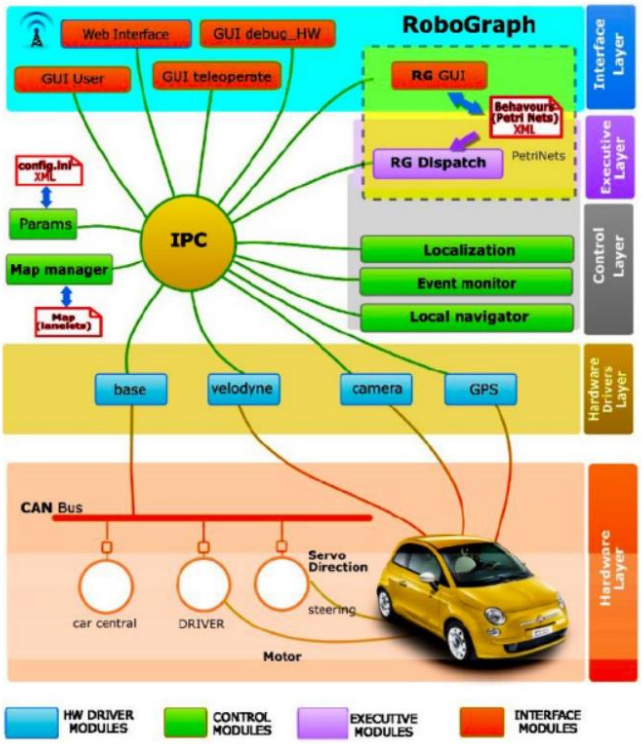

Ilustración 25 Arquitectura de navegación autónoma

A través de *rosbags* (herramientas que permiten guardar la información de los topics de un robot para poder reproducirlas en otro momento) del vehículo autónomo grabados en un entorno real, podemos verificar el funcionamiento de los algoritmos en tiempo real, sin necesidad de simulaciones y trabajando con datos reales de los sensores.

3.3.5.1. Sensores

La parte que más interesa de este vehículo autónomo a este proyecto Fin de Máster es la de los sensores, puesto que son los que van a dotar de la información necesaria a los algoritmos seleccionados, para que estos sean capaces de generar mapas y resolver el problema del SLAM.

Como ya se ha comentado en capítulos anteriores, el vehículo está dotado de un sensor láser de tipo Velodyne de 16 haces situado en la parte superior del coche, un GPS TopCon HiperPro DGPS-RTK y una cámara de stereo visión a color.

Los vehículos autónomos requieren de más sensores para navegar de una manera segura y correcta, pero para lo que atañe a este TFM, los únicos sensores que serán necesarios para completar los objetivos propuestos serán el sensor láser y el GPS.

3.3.5.1.1. Sensor láser

Los sensores láser [30] son aquellos que transmiten un paquete de ondas de presión ultrasónicas normalmente a unos 40 KHz y esperan recibir su eco. La distancia (d) al objeto que devuelve el eco se puede calcular en base a la propagación del sonido (c) y el tiempo de vuelo (t).

$$d = \frac{c * t}{2}$$

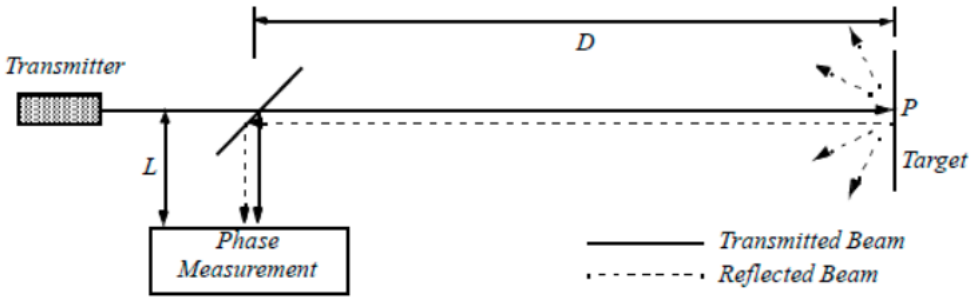

Ilustración 26 Medidor de distancia láser

Un láser consiste en un foco emisor de haces de rayos láser infrarrojos, y de una lente receptora infrarroja capaz de ver esos haces. Es decir, los rayos de luz que emite el láser rebotan contra los objetos y se reflejan sobre la lente.

El láser obtiene una nube de puntos del entorno desde diferentes ángulos, con la que el usuario podrá procesar una imagen tridimensional en tiempo real donde la posición en el espacio es precisa y la distancia que hay hasta los objetos también lo

es. En la Ilustración 27 se puede observar un ejemplo de una reconstrucción 3D del entorno gracias a la función del láser 3D.

Ilustración 27 Reconstrucción 3D del entorno con láser 3D

Existen varios modelos de láser, pero en lo que respecta al vehículo del grupo de investigación, es un láser que gira 360 grados sobre sí mismo para cubrir todo el entorno y cuya lente emite 16 rayos láser. Cada capa láser es un canal que emite una señal que genera una línea de contorno. Cuando se juntan todas esas líneas de contorno crean una reconstrucción 3D del entorno.

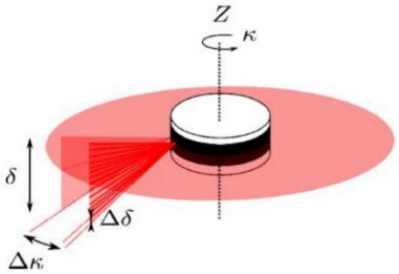

Ilustración 28 Sistema láser

Además, el láser del proyecto Teach4AgeCar tiene una resolución angular vertical de 2 grados, una resolución angular horizontal de entre 0.1 y 0.4 grados, una precisión de 3 cm y una apertura en vertical de 30 grados. El modelo exacto es el VelodyneLidarPuck (VLP16).

Ilustración 29 VelodyneLidarPuck (VLP16)

3.3.5.1.2. Sensor GPS

Un sensor GPS [31] es un sistema que permite determinar en toda la tierra la posición de cualquier objeto, desde personas hasta vehículos, con una precisión de hasta centímetros. Fue desarrollado por el Departamento de Defensa de Estados Unidos para uso militar, aunque recientemente es accesible para aplicaciones comerciales.

El GPS [30] funciona a través de una red de como mínimo 24 satélites que orbitan sobre la tierra cada 12 horas a una altura de 20.190 Km. Cuando se desea obtener la posición tridimensional, el receptor localiza automáticamente como mínimo 4 satélites de la red que indican a través de una señales su posición y hora actual. Partiendo de estas señales, el aparato sincroniza su propio reloj con el tiempo del GPS y calcula cuánto tiempo tardan en llegar las señales al equipo, y de ese modo mide la distancia al satélite. Utilizando el método de la trilateración inversa computa su propia posición.

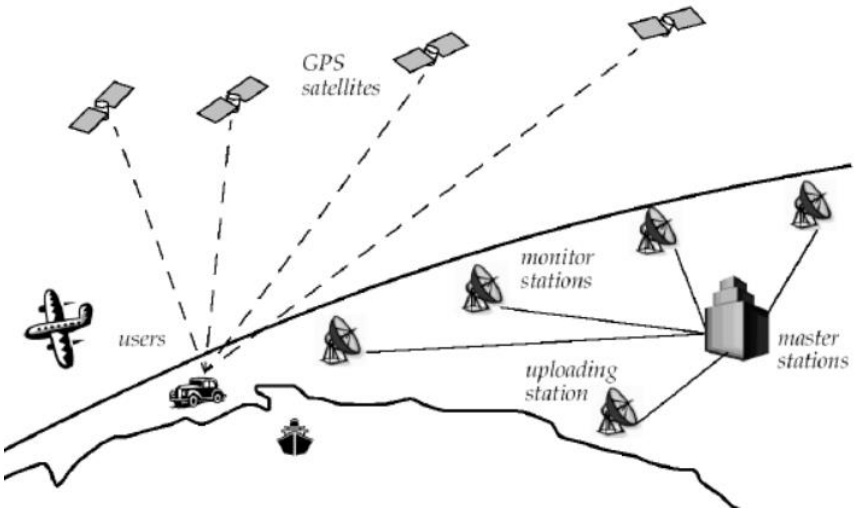

Ilustración 30 Funcionamiento sistema GPS

En la actualidad el sistema de localización del vehículo del grupo Robesafe se lleva a cabo a través del sistema GPS, más concretamente utilizan el modelo Topcon Hiper Pro-GPS.

Este modelo tiene un tiempo de operación de más de 14 horas, consume una potencia inferior a 4.2 W y tiene 20 canales de comunicación con puertos por bluetooth.

Por último, en la Ilustración 31 se pueden ver todos los sensores reales utilizados en el vehículo autónomo del grupo Robesafe para el proyecto Teach4AgeCar.

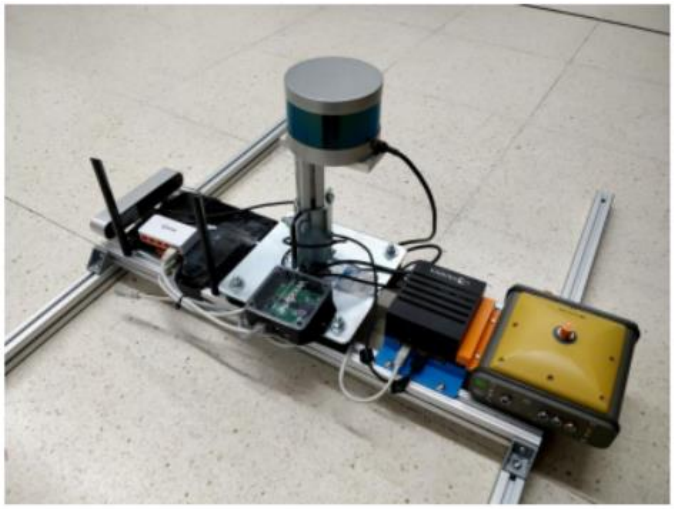

Ilustración 31 Sistema de sensores del proyecto Teach4AgeCar

Para ofrecer una visión más concreta, en la Ilustración 32 se pueden contemplar la orientación de los dos sensores descritos en este capítulo sobre el vehículo.

Ilustración 32 Orientación de los sensores Vehículo autónomo Robesafe

3.3.5.2. Sistema de localización actual

El sistema de localización de vehículo [32] se encarga de obtener la posición de este dentro de un mapa con una precisión centimétrica y en tiempo real.

Para ello se emplea un filtro de Kalman que fusiona las posiciones obtenidas por un sistema RTK y por la odometría basada en encoders del vehículo.

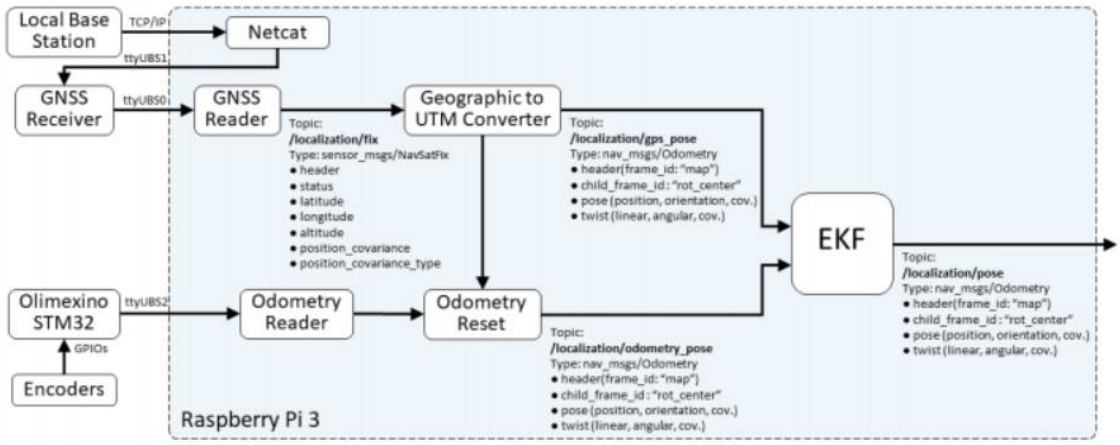

Ilustración 33 Esquema del módulo de Localización

El receptor DGNSS de la estación base, se encarga de generar correcciones diferenciales para mejorar la precisión del receptor montado sobre el vehículo. Esta información es recibida a una frecuencia de 10 Hz, a través de un router 4G.

Por otra parte, la odometría se obtiene mediante dos encoders ubicados en las ruedas traseras del vehículo conociendo el desplazamiento de cada una de ellas. Cuando la información GPS está disponible los datos de odometría son reiniciados para eliminar sus derivas. Cuando el módulo GPS pierde conexión con los satélites la odometría es capaz de proporcionar información de posición del vehículo desde el último punto válido.

A través del paquete de ROS *robot_localization* [33] se fusionan las posiciones obtenidas a partir del DGNSS y de la odometría. Mediante un Filtro de Kalman Extendido, este paquete se encargará de obtener los datos de los sensores y de estimar la posición del vehículo.

Capítulo 4.
Desarrollo.

En este capítulo se van a explicar en mayor profundidad los algoritmos de los métodos seleccionados para ser implementados en el vehículo autónomo del grupo Robesafe.

4.1. Lidar 3D SLAM con Google Cartographer

Una vez introducido el método Google Cartographer, se procede a la documentación de la parte más técnica, es decir el algoritmo exacto que usa el método para poder llevar a cabo la localización del vehículo y el mapeado del entorno.

4.1.1. Algoritmo

4.1.1.1. Local SLAM

El sistema de Google Cartographer combina un sistema "local SLAM" con otro de "global SLAM" para solucionar el problema del SLAM 2D. Ambos enfoques optimizan la posición ($\xi = (\xi x, \xi y, \xi\theta)$) donde x e y son la translación y ϴ la rotación del sensor láser. La IMU se utiliza para estimar la orientación de la gravedad, ya que se puede dar que el láser no pueda ir paralelo al suelo al desplazarse.

La posición del marco del escaneo (*scan frame*) en el marco del submapa es representada por la transformación T_ξ que transforma los puntos de los escaneos del *scan frame* en el marco de referencia del submapa, del siguiente modo.

$$T\xi p = \begin{pmatrix} cos\xi\theta & -sin\xi\theta \\ sin\xi\theta & cos\xi\theta \end{pmatrix} p + \begin{pmatrix} \xi x \\ \xi y \end{pmatrix}$$

Los submapas toman la forma de rejillas de probabilidad que se asignan a partir de puntos de cuadricula discretos dada una resolución, r.

La Ilustración 34 muestra el esquema básico de funcionamiento del algoritmo utilizado por Google Cartographer, tanto la parte de SLAM local como la de SLAM global.

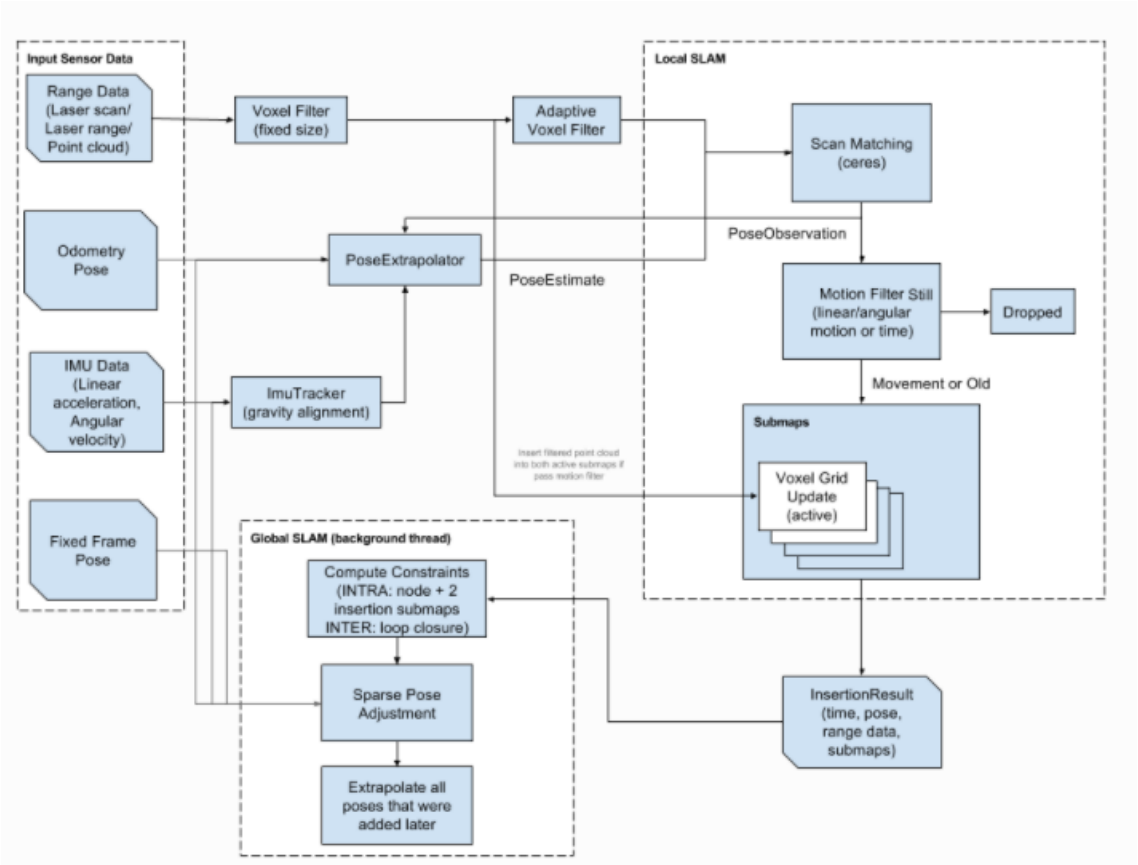

Ilustración 34 Esquema básico Local SLAM Google Cartographer

El sistema de SLAM local inserta un nuevo escaneo del sensor láser en su construcción del submapa actual mediante la coincidencia de los escaneos láser, utilizando una suposición inicial obtenida por el extrapolador de posición (*pose extrapolator*). La idea de este extrapolador es usar los datos de otros sensores para predecir dónde se debe insertar el nuevo escaneo en el submapa que se está construyendo.

Existen dos estrategias de coincidencia (*matching*) de los escaneos que el usuario puede seleccionar modificando los parámetros del algoritmo:

- **CeresScanMatcher**: Encuentra el mejor lugar donde la coincidencia de escaneo se ajuste al submapa.

  Es un proceso rápido, pero no puede corregir errores que son significativamente mayores que la resolución de los submapas. Se suele utilizar en el caso de que la configuración y el tiempo del sensor sean razonables.
- **RealTimeCorrelativeScanMatcher**: Se puede habilitar si no hay otros sensores o que no confieren mucha confianza. Es un comparador costoso y que anulará cualquier señal del resto de sensores excepto el del láser, aunque es robusto en entornos ricos en características.

El *CeresScanMatcher* se puede configurar para dar un cierto peso a cada una de sus entradas. El peso es una medida de confianza en los datos. Cuanto mayor sea el peso, mayor énfasis pondrá Cartographer en esa fuente de datos.

Para evitar insertar demasiados escaneos por submapa, una vez que el comparador de escaneos encuentra una semejanza, este pasa por un filtro de movimiento. Un escaneo se descarga si el movimiento que lo generó no es demasiado significativo. El escaneo será insertado, con la condición indispensable de que el movimiento esté por encima de una cierta distancia, ángulo o umbral del tiempo. Algunos de los parámetros son:

- TRAJECTORY_BUILDER_nD.motion_filter.max_time_seconds: restricción de tiempo.
- TRAJECTORY_BUILDER_nD.motion_filter.max_distance_meters: restricción de distancia.
- TRAJECTORY_BUILDER_nD.motion_filter.max_angle_radians: restricción de ángulo.

Un submapa se considera completo cuando el SLAM local ha recibido una determinada cantidad de datos del láser. El SLAM local se desvía con el tiempo, y el SLAM global debe corregir esa desviación. Los submapas deben ser lo suficientemente pequeños como para que la deriva dentro de ellos esté por debajo de la resolución, pero a la vez lo suficientemente grandes para que el cierre del bucle funcione correctamente. El parámetro que indica el tamaño que tendrán los submapas es:

- TRAJECTORY_BUILDER_nD.submaps.num_range_data

4.1.1.2. Global SLAM

El sistema de SLAM global tiene como función reorganizar los submapas entre sí para que formen un mapa global coherente. Es decir, se encargar de modificar la trayectoria actual construida, para poder alinear correctamente los submapas respecto a los cierres de bucles.

Esta optimización se ejecuta en lotes una vez se haya insertado un cierto número de nodos de trayectoria. Dependiendo de la frecuencia que se necesite para ejecutar esos lotes, se puede ajustar el tamaño de estos. El parámetro que se encarga de ello es:

- POSE_GRAPH.optimize_every_n_nodes

Global SLAM es esencialmente una optimización del grafo de posición que funciona construyendo restricciones entre nodos y submapas y después optimizando el grafo de restricciones resultantes. Las restricciones son como pequeñas "cuerdas" que unen todos los nodos entre sí. La red resultante es el grafo de posición (*pose graph*).

Para limitar la cantidad de restricciones, Cartographer sólo considera un conjunto submuestreado de todos los nodos cercanos para la construcción de restricciones. Se controla a través de una constante de proporción del muestreo (*sampling ratio*).

- POSE_GRAPH.constraint_builder.sampling_ratio

Cartographer implementa un comparador de escaneo llamado *FastCorrelativeScanMatcher*, que hace posible la coincidencia de escaneo de cierres de bucles en tiempo real. Elimina de una manera eficiente las coincidencias y trabaja con diferentes resoluciones.

- POSE_GRAPH.constraint_builder.fast_correlative_scan_matcher_3d.branch_and bound_depth: profundidad del árbol de exploración (mecanismo de bifurcación y enlace), método *FastCorrelativeScanMatcher*.
- POSE_GRAPH.constraint_builder.fast_correlative_scan_matcher_3d.full_resoluti on_depth: resolución de las cuadriculas para eliminar las coincidencias incorrectas, método *FastCorrelativeScanMatcher*.

Además, existen otros parámetros a los que se les puede introducir un peso determinado, tal y como se explicó en la parte de SLAM local. Entre ellos se pueden destacar: las restricciones globales (cierres de lazos), las restricciones no globales (matcher), las medidas de aceleración y rotación de la IMU, las estimaciones de la posición aproximada en SLAM Local, una fuente de odometría etc.

- POSE_GRAPH.constraint_builder.loop_closure_translation_weight: peso de moviento de traslación en cierre de lazo.
- POSE_GRAPH.constraint_builder.loop_closure_rotation_weight: peso de rotación en cierre de lazo.
- POSE_GRAPH.matcher_translation_weight: restricción no global del movimiento de traslación.
- POSE_GRAPH.matcher_rotation_weight: restricción no global de rotación.
- POSE_GRAPH.optimization_problem.*_weight
- POSE_GRAPH.optimization_problem.ceres_solver_options

4.1.2. Topics y Parámetros

Google Cartographer ofrece la posibilidad de modificar cada uno de sus archivos fuente y todos sus parámetros para poder ajustarlo a la aplicación que el usuario desee.

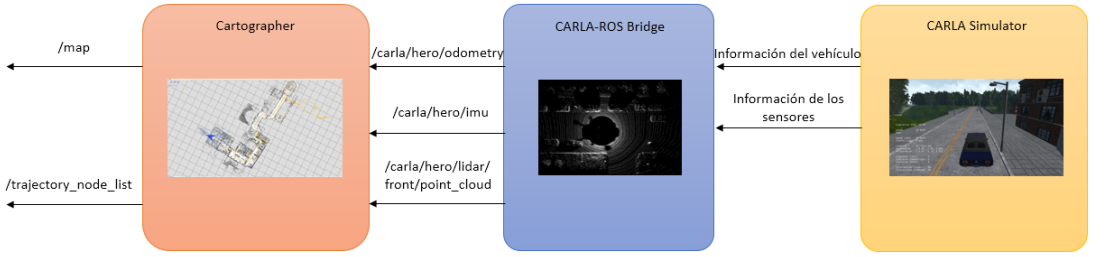

Ilustración 35 Esquema básico TFM Cartographer

En el caso de este TFM se parte de una simulación en Carla Simulator, por lo que se debe hacer una lectura previa de la información que ofrecen los sensores para luego poder enviarla, a través de *Carla-ROS-Bridge*, a Google Cartographer y que este sea capaz de generar un mapa con las especificaciones propuestas.

Como se puede observar en Ilustración 35, los tres topics fundamentales necesarios como entrada a Cartographer son:

Subscriber:

- **/carla/hero/odometry**:

  En Cartographer el uso de la odometría es opcional tanto para 3D como para 2D, pero en el caso de este TFM el topic de odometría será una de las entradas del algoritmo, ya que el vehículo Techs4AgeCar dispone de odometría en las ruedas traseras.

  El uso de la odometría permite conocer el recorrido seguido por el vehículo, de forma que simplifica el proceso de vinculación entre nubes de puntos aportando una información aproximada de su posición. Por lo tanto, la odometría participara en el proceso del Global SLAM.

- **/carla/hero/imu**:

  Al intentar realizar la optimización global, *CeresScanMatcher* intenta mejorar la posición entre su IMU y los sensores de rango. Una adquisición bien elegida con muchas restricciones de cierre de lazo puede mejorar la calidad de esas correcciones y convertirse en una fuente fiable de corrección de posición.

  Si queremos utilizar Google Cartographer en 3D es necesario una IMU, principalmente para medir la gravedad. La gravedad es necesaria por dos razones:

  1. Para definir la dirección z y poder alinear correctamente el la trayectoria y el mapa resultante.
  2. El balanceo y el cabeceo se pueden derivar bastante bien de las lecturas de la IMU una vez se ha establecido la dirección de la gravedad. Esto ahorra trabajo al comparador de escaneo al reducir la ventana de búsqueda.

  Como el objetivo principal de este TFM es conseguir aplicar una técnica de localización y mapeado simultaneo en 3D, el topic de la IMU será una de las entradas en Cartographer por las razones descritas anteriormente.

- **/carla/hero/lidar/front/point_cloud**:

  La nube de puntos es la parte más esencial para el correcto funcionamiento del algoritmo. La información del láser es utilizada por Cartographer para poder construir el mapa bidimensional o tridimensional progresivamente.

  Con la nube de puntos se forman los submapas (en forma de rejillas de ocupación 2D) y con la unión de esos submapas, lazos cerrados que luego constituyen un mapa del entorno.

A parte de los topics que se han utilizado para realizar las pruebas con Google Cartographer, existe una gran variedad de ellos con los que el usuario puede conseguir obtener la información que sea necesaria para el correcto funcionamiento de su aplicación.

Publisher:

- **/constraint_list [visualization_msgs/MarkerArray]** 1 publisher
  Lista de restricciones entre nodos y submapas.
- **/submap_list [cartographer_ros_msgs/SubmapList]** 1 publisher
  Genera una lista de submapas, que contiene información sobre:
  - **int32 submap_index**: índice del submapa.
  - **int32 submap_version**: la versión del submapa.
  - **geometry_msgs/Pose pose**: la posición donde se encuentra el submapa.
- **/scan_matched_points2 [sensor_msgs/PointCloud2**] 1 publisher
  Nube de puntos de entrada del sistema.
- **/tf [tf2_msgs/TFMessage]** 2 publishers
  Información sobre la relación de transformación entre los distintos sistemas de coordenadas.
- **/map [nav_msgs/OccupancyGrid]** 1 publisher
  Topic que almacena el mapa generado por el algoritmo.
- **/trajectory_node_list [visualization_msgs/MarkerArray]** 1 publisher
  Crea una lista de los puntos que ha ido recorriendo el vehículo.

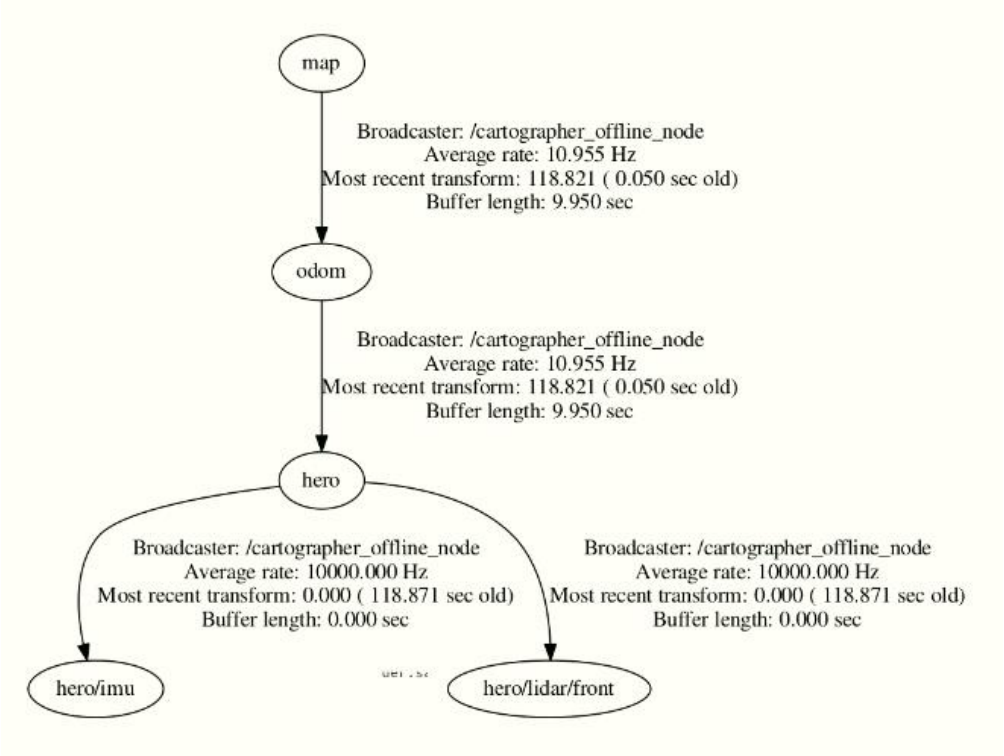

Ilustración 36 Árbol de transformadas Cartographer

La Ilustración 36 muestra el árbol de transformadas que quedaría al hacer funcionar el algoritmo con los topics que se han extraído de Carla Simulator.

Es necesario ajustar los parámetros preestablecidos de Google Cartographer para que funcionen según los requisitos de la aplicación.

Los parámetros fundamentales que han marcado la diferencia al ponerles un valor distinto al predeterminado son:

- **Provide_odom_frame=true**

  Si está activada, la posición local será publicada como *odom_frame* en el *map_frame*.
- **Use_odometry=true**

  Si está activada, la odometría será utilizada por el algoritmo, por lo que el topic de la odometría deberá referenciarse al odom de Cartographer.
- **Num_point_clouds=1**

  Por defecto está puesto a 2 pero para este TFM el valor se cambiará a 1, ya que solo existe una nube de puntos en la simulación empleada.

- **TRAJECTORY_BUILDER_3D.num_accumulated_range_data = 4**

  Se recomienda proporcionar tantos datos de rango (mensajes de ROS) por escaneo como sea posible, ya que luego ese escaneo se utilizará para compararlo con el siguiente y saber si hay coincidencias.

  En este caso el sensor lidar no necesita muchos datos de rango ya que con un solo mensaje recibe una nube de puntos completa.
- **POSE_GRAPH.optimize_every_n_nodes = 1**

  Como ya se explicó, este parámetro sirve para la optimización del cierre de lazos. Si su valor es de 0, entonces Cartographer se centrará únicamente en el Local SLAM y dejará de lado el Global SLAM.

  Por defecto el valor era de 90, sin embargo, para este proyecto se ha bajado hasta 1.
- **POSE_GRAPH.constraint_builder.sampling_ratio =1**

  Cuanto mayor sea el número de este parámetro, considerará nodos más lejanos para la construcción de restricciones.

  Por defecto a 0.03 pero para este TFM a 1, para ampliar la búsqueda de nodos.
- **POSE_GRAPH.constraint_builder.log_matches**

  Es el radio de búsqueda del cierre de lazos. Si el lazo es muy grande y se ha acumulado mucho error, podría darse el caso de que no se cerrara el lazo correctamente.

  Por defecto su valor es de 15 pero para este TFM se aumenta hasta 90, para que sea más fácil buscar en zonas más alejadas.
- **use_online_correlative_scan_matching=false**

  Al poner su valor a false, el método de optimización utilizado será *CeresScanMatching*, que no inhibe las señales del resto de sensores.

El resto de los parámetros de los ficheros de Google Cartographer no han sido modificados y todos sus valores han quedado marcados como los de defecto.

4.1.3. Resultados con CARLA Simulator

Cuando las comunicaciones entre CARLA Simulator, *CARLA-ROS-bridge* y el algoritmo Google Cartographer están establecidas y los parámetros han sido modificados para que casen con los requisitos del proyecto, se puede proceder a la visualización en la plataforma RViz del mapa resultante.

CARLA dispone de varias ciudades con características diferentes entre ellas. Para este proyecto se selecciona la ciudad 02, con elementos del entorno muy marcados y calles estrechas, que permiten que el algoritmo pueda generar un mapa más fácilmente.

En CARLA se genera un recorrido con uno de los vehículos autónomos, alrededor de una manzana de la ciudad (ver la Ilustración 37). Como uno de los objetivos es ver si Global SLAM consigue cuadrar bien todos los submapas, el vehículo dará dos vueltas a la manzana.

Ilustración 37 Recorrido CARLA Simulator

El resultado aplicando ese mismo recorrido y utilizando Cartographer es el siguiente:

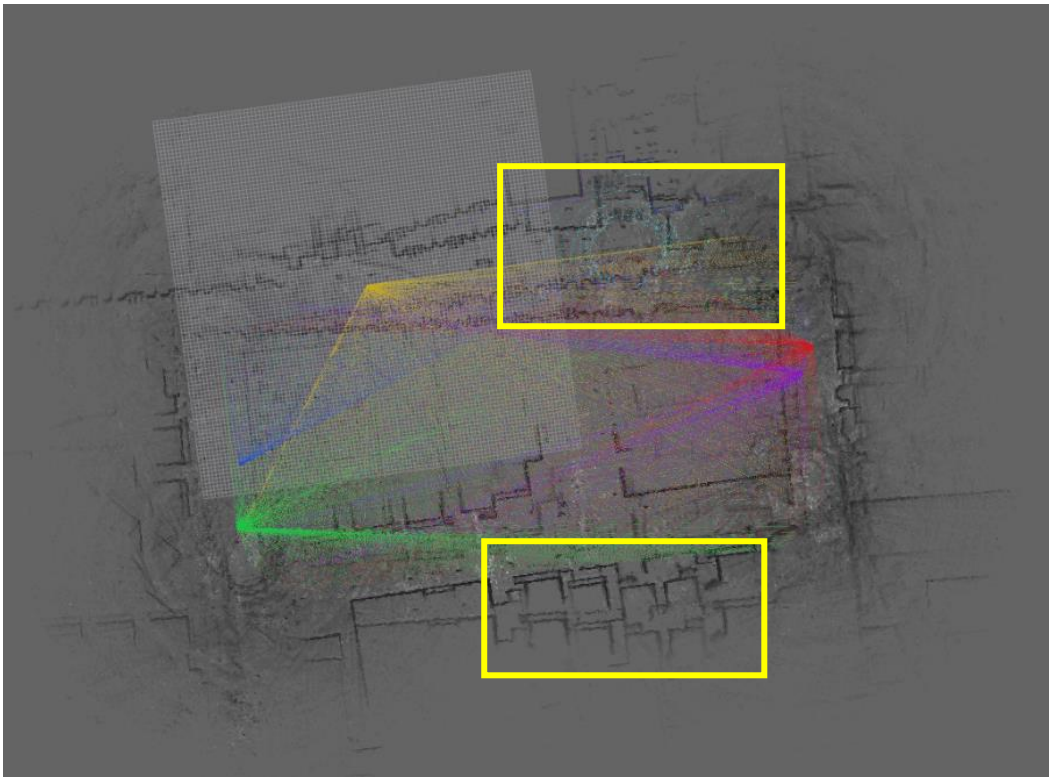

Ilustración 38 Recorrido Google Cartographer

En los dos cuadrados amarillos sobre la Ilustración 38 se muestran las incoherencias del cierre de lazos una vez finalizados los dos recorridos.

Global SLAM no alinea bien los submapas generados en el transcurso del recorrido, y por lo tanto superpone unos encima de otros con visibles desviaciones.

A pesar de modificar los parámetros del algoritmo, no se ha conseguido un resultado diferente del expuesto en la Ilustración 38. La conclusión una vez finalizado todo el periodo de modificaciones es que, el algoritmo no está preparado para entornos en exterior que no tengan muchos elementos que generen buenas coincidencias. Google Cartographer funciona muy bien en entornos de interior.

Además, a pesar de estar implementando Cartographer en su formato 3D, ya que utiliza un lidar 3D como fuente de información, el mapa resultante es un mapa 2D formado por los submapas locales (rejillas de ocupación) relacionados por un grafo de posiciones para darles consistencia global (aunque en exteriores esto no se consigue, como se muestra en esta aplicación).

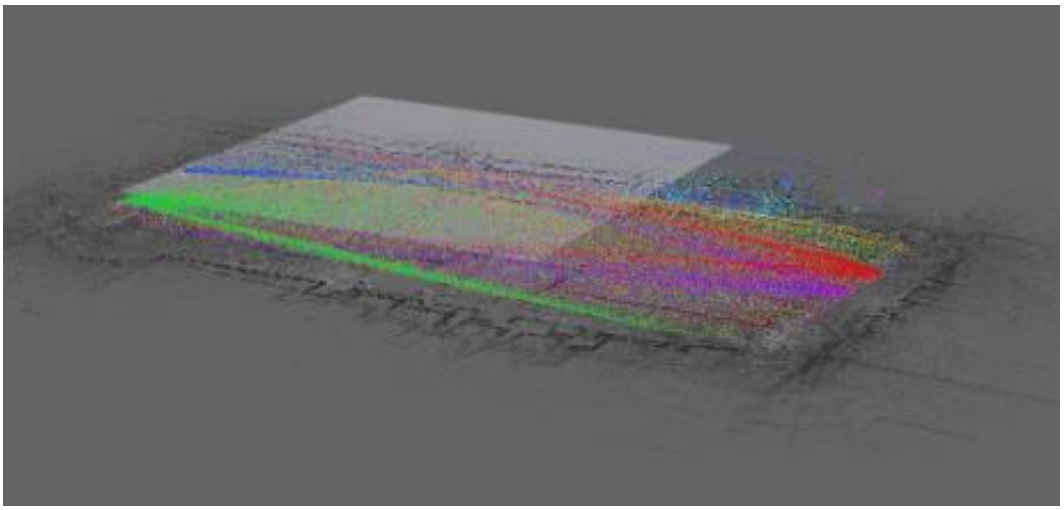

Ilustración 39 Recorrido en perspectiva Google Cartographer

4.2. Lidar 3D SLAM con Lidar Odometry and Mapping (LOAM)

4.2.1. Algoritmo

Una vez introducido el método LOAM, se procede a la documentación de la parte más técnica, es decir el algoritmo exacto que usa el método para poder llevar a cabo la localización del vehículo y el mapeado del entorno.

4.2.1.1. Odometría láser

La información láser tiene una resolución de 0.25˚ en un plano. Al girar a una velocidad de 180˚/s y generar información a 40Hz, la resolución vertical es de 4,5˚. Por ello, se extrae la información de planos paralelos.

Los puntos de un escaneo láser son ordenados según su pertenencia a regiones de bordes o planas. Para obtener una representación del entorno, el escaneo se divide en cuatro subregiones. Cada subregión puede proporcionar 2 puntos de borde y cuatro puntos de plano.

A la hora de seleccionar los puntos característicos, se evita escoger puntos cuyos puntos de alrededor ya estén seleccionados o algunos en los que su superficie planar esté paralela a los rayos láser (Ilustración 40 (a), punto b). Por lo general estos puntos se consideran poco fiables.

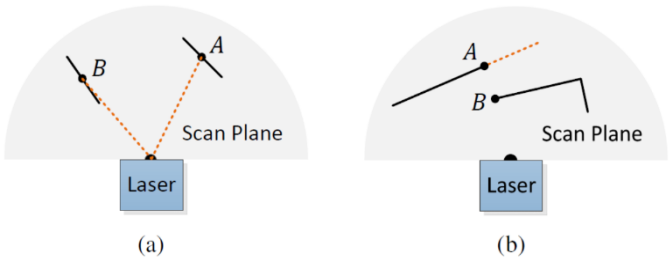

Ilustración 40 Puntos característicos LOAM

El punto A de la Ilustración 40 (b) es otro ejemplo de punto que se debe evitar, ya que se encuentra en una zona de oclusión al estar su superficie bloqueada por otro objeto delante suyo. De todas formas, si el láser se moviera, el punto A dejaría de estar en una zona de oclusión y podría verse.

El algoritmo estima el movimiento del láser dentro de un barrido. Al finalizar el barrido, la nube de puntos percibida se reproyecta un periodo de tiempo después, al que desarrolló su barrido.

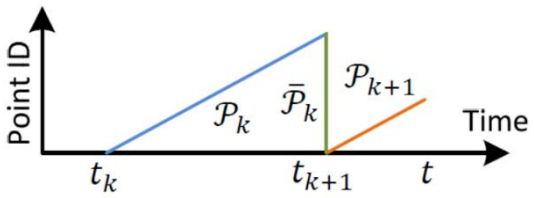

Ilustración 41 Desarrollo de un barrido láser, Google Cartographer

El comienzo de un barrido con una nube de puntos (P_{k+1}), crece durante el transcurso del barrido a medida que más puntos se perciben. La odometría láser estima los 6 grados de libertad del movimiento durante el barrido y gradualmente incluye más puntos a medida que crece la nube de puntos.

El algoritmo de odometría toma como entradas la nube de puntos del instante anterior, la nube de puntos actual y la transformada de la posición.

4.2.1.2. Mapeado láser

El algoritmo de mapeado se ejecuta a frecuencias más bajas que el algoritmo encargado de la odometría y únicamente será ejecutado una vez comience cada barrido.

Al final del barrido, la odometría láser genera una nube de puntos libre de distorsiones y simultáneamente la transformada de la posición que contiene el movimiento del láser durante el barrido. El algoritmo registra la nueva nube de puntos en las coordenadas del mundo.

Se divide la nube de puntos en rejillas cubicas de 10 metros para buscar puntos característicos, y el algoritmo se queda con los puntos que forman un borde, o los que forman un plano.

A continuación, calcula la covarianza para conocer la alineación entre el mapa y la nube de puntos (aplicando Levenberg-Manquardt para resolver la optimización no lineal).

4.2.2. Topics y Parámetros

LOAM ofrece la posibilidad de modificar cada uno de sus archivos fuente y todos sus parámetros para poder ajustarlo a la aplicación que el usuario desee.

En el caso de este TFM se parte de una simulación en Carla Simulator, por lo que se debe hacer una lectura previa de la información que ofrecen los sensores para luego poder enviarla, a través de *Carla-ROS-Bridge*, a LOAM y que este sea capaz de generar un mapa con las especificaciones propuestas.

Como se puede observar en la Ilustración 42, únicamente necesita la información de la nube de puntos (un requerimiento específico de este algoritmo)

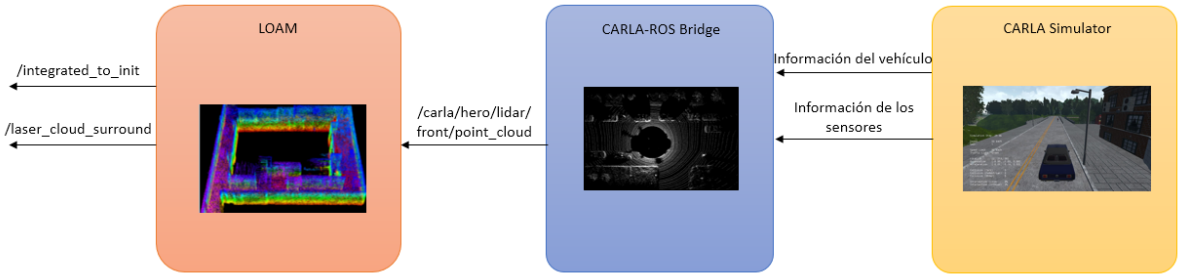

Ilustración 42 Esquema básico TFM LOAM

Subscriber:

- **/carla/hero/lidar/front/point_cloud**:

La nube de puntos es la parte esencial para el correcto funcionamiento del algoritmo. La información del láser es utilizada por LOAM para poder construir el mapa tridimensional progresivamente.

En este caso y a diferencia de Google Cartographer, LOAM no requiere un topic de la odometría para funcionar. El algoritmo es capaz de identificar elementos coincidentes entre nubes de puntos consecutivas, suficientes como para generar su unión y crear un mapa del entorno.

Publisher:

- **/localization/gps_pose [nav_msgs/Odometry]** 1 publisher
  Posición estimada por el algoritmo en metros respecto al origen.
- **/integrated_to_init [nav_msgs/Odometry]** 1 publisher
  Topic que contiene la información de odometría del vehículo a lo largo del recorrido.
- **/velodyne_points [sensor_msgs/PointCloud2]** 1 publisher
  Nube de puntos de entrada.
- **/tf [tf2_msgs/TFMessage]** 3 publishers
  Información sobre la relación de transformación entre los distintos sistemas de coordenadas.
- **/laser_cloud_surround** [sensor_msgs/PointCloud2] 1 publisher

  Mapa completo que va generando el algoritmo

La Ilustración 43 muestra el árbol de transformadas resultante con LOAM SLAM

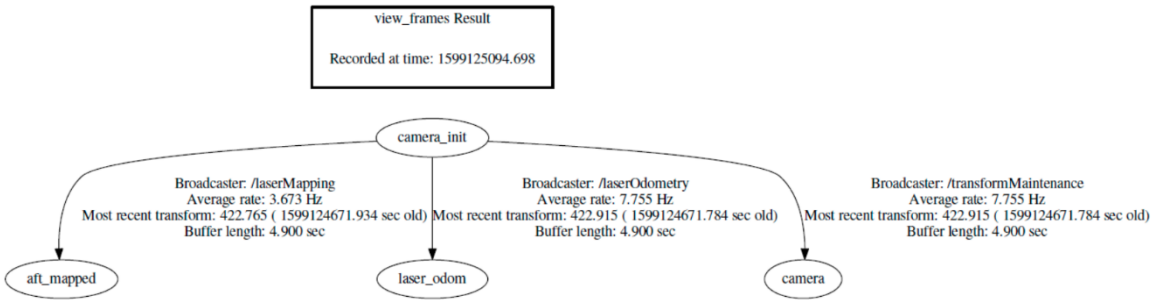

Ilustración 43 Árbol de transformadas LOAM

El algoritmo ha funcionado bien referenciando la nube de puntos de CARLA con el topic de LOAM. Algunos de los parámetros más destacables son:

- **MaxIterations**: máximo número de iteraciones por mapa.
- **DeltaTAbort**: ///< optimization abort threshold for deltaT

4.2.3. Resultados con CARLA Simulator

Cuando las comunicaciones entre CARLA Simulator, *CARLA-ROS-bridge* y el algoritmo LOAM han sido establecidas se puede proceder a la visualización en la plataforma Rviz del mapa resultante. Utilizando para ello el mismo escenario que para el resto de las ocasiones.

Ilustración 44 Recorrido CARLA Simulator

El resultado aplicando ese mismo recorrido y utilizando LOAM es:

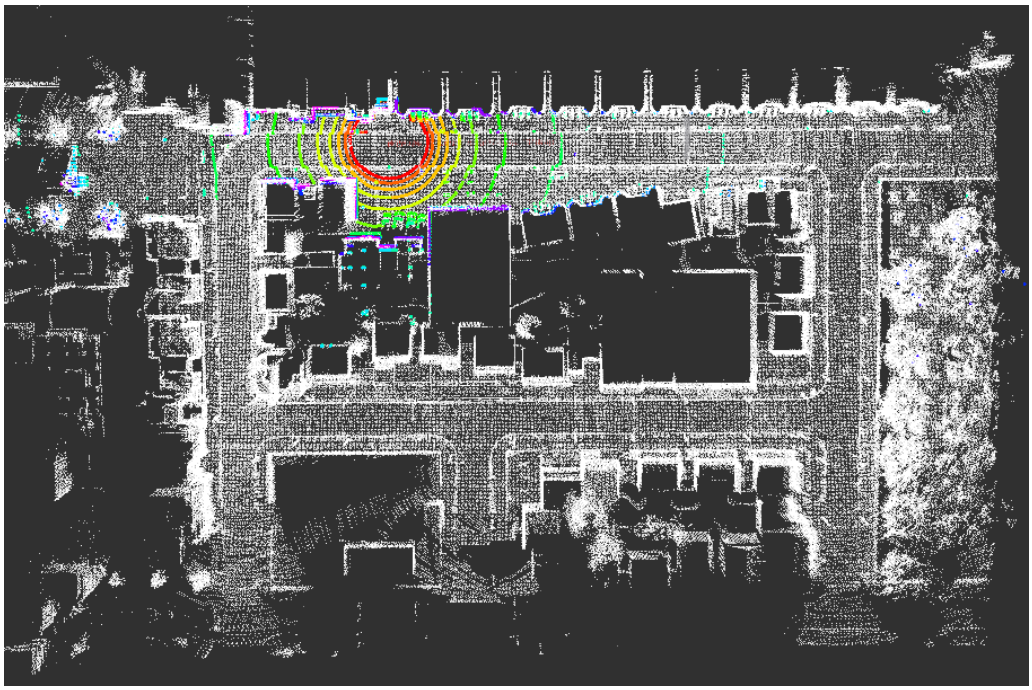

Ilustración 45 Recorrido mundo CARLA aplicando LOAM SLAM

La Ilustración 45 es prácticamente una copia de la manzana que se ve en la Ilustración 44. La nitidez del entorno y de los elementos que lo rodean (árboles, edificios y calles) reflejan el éxito del algoritmo LOAM SLAM.

El algoritmo de odometría ha sabido casar perfectamente las diferentes nubes de puntos que han ido apareciendo a lo largo de la simulación. No existe un desplazamiento significativo entre los submapas, es decir el error acumulado es corregido por el sistema de SLAM.

LOAM ha conseguido encontrar todas las coincidencias entre los puntos del entorno entre un barrido y otro, lo que genera grandes resultados.

La Ilustración 46 reitera los buenos resultados ya comentados. La imagen en perspectiva ofrece una visión muy buena de los edificios de la ciudad de CARLA, con todos sus elementos en 3D (justamente lo que se venía buscando en este TFM).

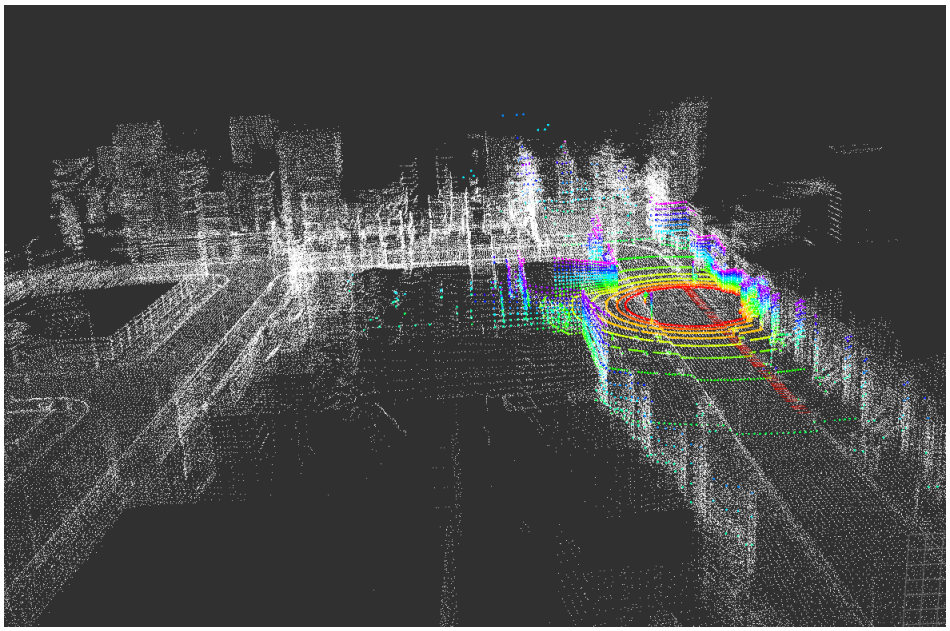

Ilustración 46 Recorrido mundo Carla aplicando LOAM SLAM vista de perspectiva

4.3. Lidar 3D SLAM con Hdl_Graph_SLAM

4.3.1. Algoritmo

Una vez introducido el método Hdl_Graph_SLAM, se procede a la documentación de la parte más técnica, es decir el algoritmo exacto que usa el método para poder llevar a cabo la localización del vehículo y el mapeado del entorno.

4.3.1.1. Algoritmo de detección de bucle

En primer lugar y en función de la longitud y la traslación de la trayectoria entre nodos, el algoritmo detecta los candidatos de bucle. A continuación, para validar los candidatos a bucle, se aplica el método de coincidencia de escaneos NDT entre los nodos de cada candidato sujeto a estudio. Si la puntuación de aptitud (*fitness score*) es inferior a un umbral, el bucle candidato es agregado al gráfico como un borde entre los nodos. Cada vez que se encuentre un bucle, el gráfico de posición es actualizado, utilizando para su optimización g2o [34].

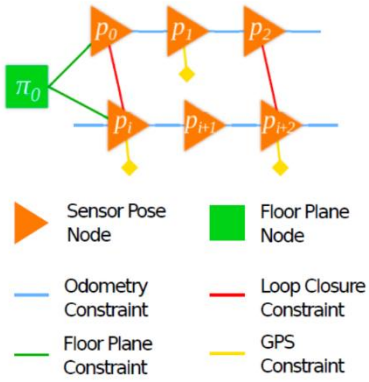

Ilustración 47 Estructura gráfico de posición

A medida que el mapa generado crece, este tiende a doblarse debido al error de rotación acumulado de las coincidencias de exploración. Para compensar el error, se introducen restricciones de posición GPS y del plano del suelo para entornos de interior y exterior respectivamente.

Restricción del plano del suelo

Para genera un mapa fiable del entorno interior, el algoritmo asume la existencia de un solo plano del suelo, es decir todo lo que en la nube de puntos parezca un suelo, Hdl_Graph_SLAM lo hace encajar con el suelo del mapa. De modo que se evitan los errores que podrían darse en el caso en el que la nube de puntos se hubiera inclinado con un posible frenazo del vehículo.

La altura del sensor se da como conocida, así como los puntos dentro de esa altura que deben contener los puntos del plano suelo. RANSAC se aplica a la nube de puntos extraída y detecta el plano del suelo. Si la normal del plano detectado es casi vertical, el algoritmo considera que el plano del suelo es correcto. Un ejemplo es la Ilustración 48.

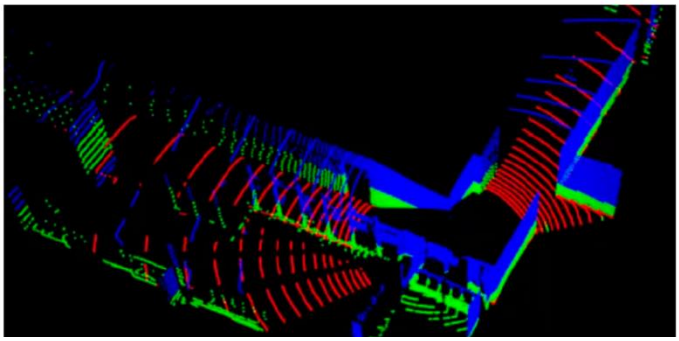

Ilustración 48 Ejemplo detección plano del suelo

Los puntos verdes corresponden a los puntos extraídos teniendo en cuenta el umbral de altura, los puntos rojos pertenecen al plano del suelo detectados a través de RANSAC. El plano del suelo es detectado cada 10 segundos y conecta los nodos de la posición del sensor con los nodos fijos del plano del suelo.

Restricción GPS

En entornos de exterior donde el suelo no es plano, el algoritmo emplea la restricción de posición del GPS, en lugar de la restricción del plano del suelo.

Cada optimización requiere una transformación previa de los datos del GPS (en coordenadas cartesianas, con este y norte) a UTM (*Universal Transverse Mercator*). A continuación, cada dato GPS tendrá un nodo de posición asociado, que a su vez está relacionado con la información de posición del nodo anterior.

El error entre el vector de traslación del nodo de posición y la posición GPS viene dado por:

$$\boldsymbol{e}_i = \boldsymbol{t}_t - \boldsymbol{T}_t.$$

Se puede estimar el movimiento del sensor iterativamente aplicando un algoritmo de coincidencia de exploración como parte del proceso de SLAM.

El método NDT que utiliza los datos de velocidad angular proporcionados por el láser 3D, es incluido en el algoritmo a través de un filtro de Kalman.

4.3.2. Topics y Parámetros

Hdl_Graph_SLAM ofrece la posibilidad de modificar cada uno de sus archivos y todos sus parámetros para poder ajustarlo a la aplicación que el usuario desee.

En el caso de este TFM se parte de una simulación en Carla Simulator, por lo que se debe hacer una lectura previa de la información que ofrecen los sensores para luego poder enviarla, a través de Carla ROS Bridge, a Google Cartographer y que este sea capaz de generar un mapa con las especificaciones propuestas.

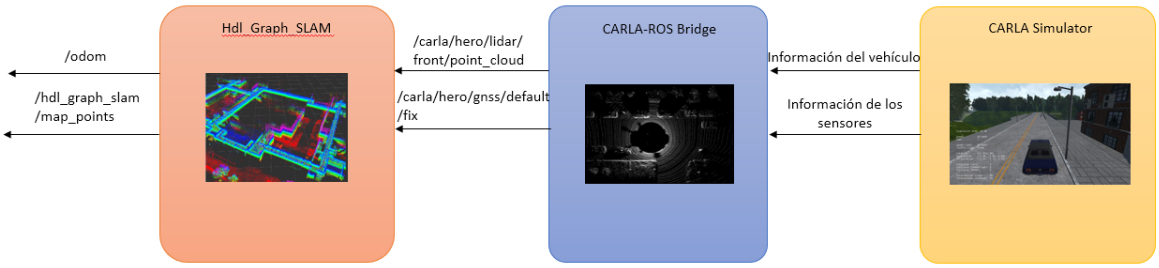

Ilustración 49 Esquema básico TFM Hdl_graph_slam

Como se puede observar en la Ilustración 49, Hdl_graph_slam únicamente necesita la información de la nube de puntos y la información del GPS.

Subscriber:

- **/carla/hero/lidar/front/point_cloud**:

  La nube de puntos es la parte esencial para el correcto funcionamiento del algoritmo. La información del láser es utilizada por Hdl_Graph_SLAM para poder construir el mapa tridimensional progresivamente.

  En el caso real, donde los datos de la nube de puntos son obtenidos a través del vehículo autónomo del grupo Robesafe, el nombre del topic será /velodyne_points.

- **/carla/hero/gnss/default/fix**:

  Para poder generar la restricción GPS en entornos de exterior cuyo suelo no sea plano, debemos obtener a través CARLA-Ros-Bridge la información del sensor GPS del vehículo autónomo.

  Al disponer de datos proporcionados por GPS, el algoritmo no necesita aplicar odometría para obtener la posición del coche dentro del mapa.

  El mensaje del GPS es de tipo NavSatFix que requiere que los datos vengan dados en latitud y longitud.

  Como el coche real fusiona sistemas de GPS y odometría, este conoce su posición en metros respecto al origen de un mapa local. Para conocer la latitud y longitud se necesita conocer la posición del mapa y combinar los dos tipos de datos en metros antes de pasarlo de nuevo a latitud y longitud.

A parte de los topics que se han utilizado para realizar las pruebas con Hdl_Graph_SLAM, existe una gran variedad de ellos con los que el usuario puede conseguir obtener la información que sea necesaria para el correcto funcionamiento de su aplicación.

Publisher (Todo topic que está siendo publicado):

- **/velodyne_nodelet_manager/bond [bond/Status]** 4 publishers
  Topic de generación entre nodos.
- **/hdl_graph_slam/map_points [sensor_msgs/PointCloud2]** 1 publisher
  Mapa generado por el algoritmo.
- **/tf [tf2_msgs/TFMessage]** 3 publishers
  Información sobre la relación de transformación entre los distintos sistemas de coordenadas.
- **/odom [nav_msgs/Odometry]** 1 publisher
  Información sobre la odometría del vehículo.
- **/scan_matching_odometry/read_until [std_msgs/Header]** 1 publisher
  Se encarga de las coincidencias generadas por la odometría.
- **/filtered_points [sensor_msgs/PointCloud2]** 1 publisher
  Son en cada momento los puntos que extrae el algoritmo de la nube de entrada disponible, para añadir al mapa.
- **/floor_detection/floor_coeffs [hdl_graph_slam/FloorCoeffs]** 1 subscriber
  Este tipo de restricción optimiza el grafico para que los planos del suelo siempre contengan los mismos nodos de posicion. Esta diseñado para reducir el error de rotacion provocado por la coincidencia de escaneos en entornos de interior.
- **/gpsimu_driver/imu_data [sensor_msgs/Imu]** 1 subscriber
  Esta restricción rota cada nodo de posicion para que el vector de aceleracion asociado con el nodo se vuelva vertical (como el nodo de gravedad). Es util para comensar los errores de rotacion de inclinacion acumulados por la coincidencia de los escaneos. No es necesario darle mucho peso a esta restricción por el hecho de no conocer la aceleración del movimiento

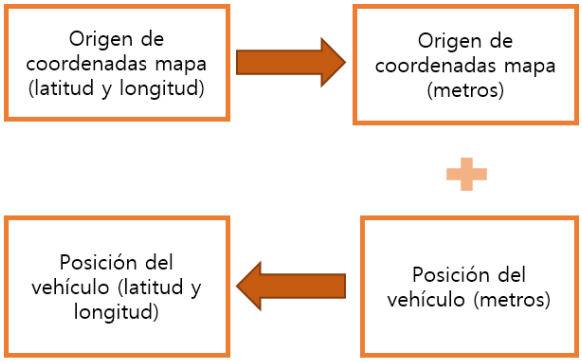

Ilustración 50 Coordenadas GPS Hdl_Graph_SLAM

El algoritmo posee múltiples parámetros configurables a gusto del usuario para generar un buen desarrollo de su aplicación.

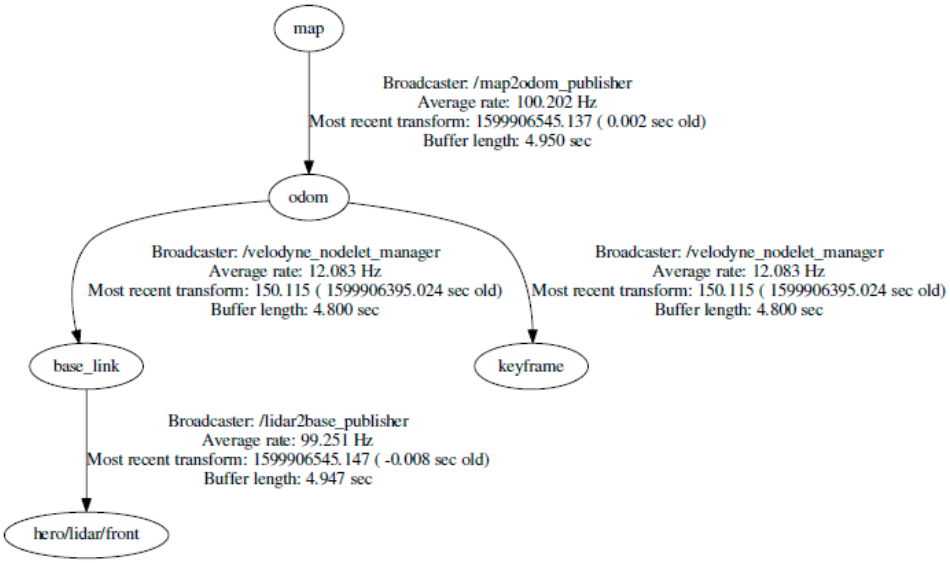

Ilustración 51 Árbol de transformadas Hdl_Graph_SLAM

La Ilustración 51Ilustración 43 muestra el árbol de transformadas resultante con Hdl_Graph_SLAM.

El algoritmo ha funcionado bien referenciando la nube de puntos de CARLA con los topics de Hdl_Graph_SLAM. Alguno de los parámetros más destacables es:

- Para configurar las desviaciones estándar del GPS y de la IMU:
  - **gps_edge_stddev_xy**
  - **gps_edge_stddev_z**
  - **imu_orientation_edge_stddev**
  - **imu_acceleration_edge_stddev**

- **distance_thresh**: La distancia alrededor de la posición estimada en la que busca solapamientos para cierre de lazo.
- **fitness_score_thresh**: Umbral de coincidencia entre dos posiciones para considerar cierre de lazo.

4.3.3. Resultados con CARLA Simulator

Cuando las comunicaciones entre CARLA Simulator, *CARLA-ROS-bridge* y el algoritmo Hdl_Graph_SLAM han sido establecidas se puede proceder a la visualización en la plataforma Rviz del mapa resultante. Utilizando el mismo escenario que las anteriores.

Ilustración 52 Recorrido CARLA Simulator

El resultado aplicando ese mismo recorrido y utilizando Hdl_Graph_SLAM es:

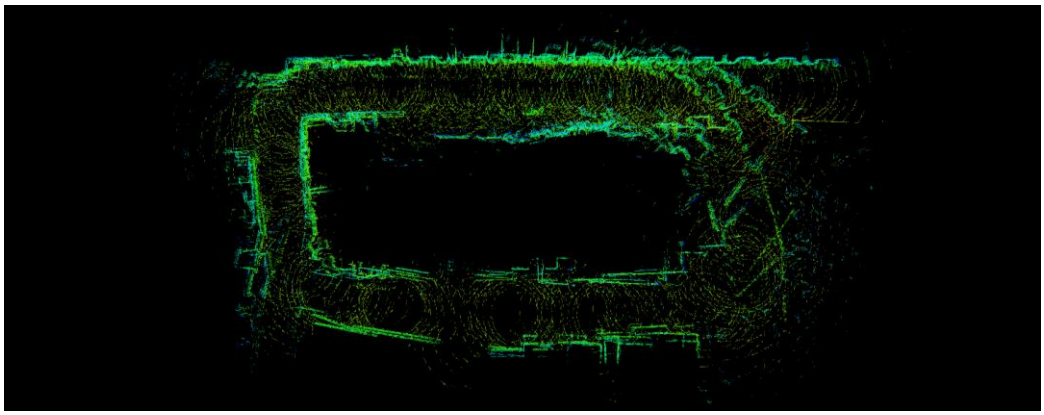

Ilustración 53 Recorrido mundo CARLA aplicando Hdl_Graph_SLAM

Aunque Hdl_Graph_SLAM produce un mapa tridimensional del entorno, no es capaz de realizar el cierre de lazo con existo como se muestra en la Ilustración 53 Recorrido mundo CARLA aplicando Hdl_Graph_SLAM pese a utilizar información GPS como restricción.

A pesar de que se parece a la Ilustración 52 el resultado no es suficientemente bueno para ser empleado en la navegación de un vehículo autónomo.

La Ilustración 54 muestra una vista en perspectiva de la mitad de la manzana recorrida. En esta sección el algoritmo es capaz de encontrar una coincidencia suficientemente válida solapando la información obtenida del entorno en las dos vueltas realizadas.

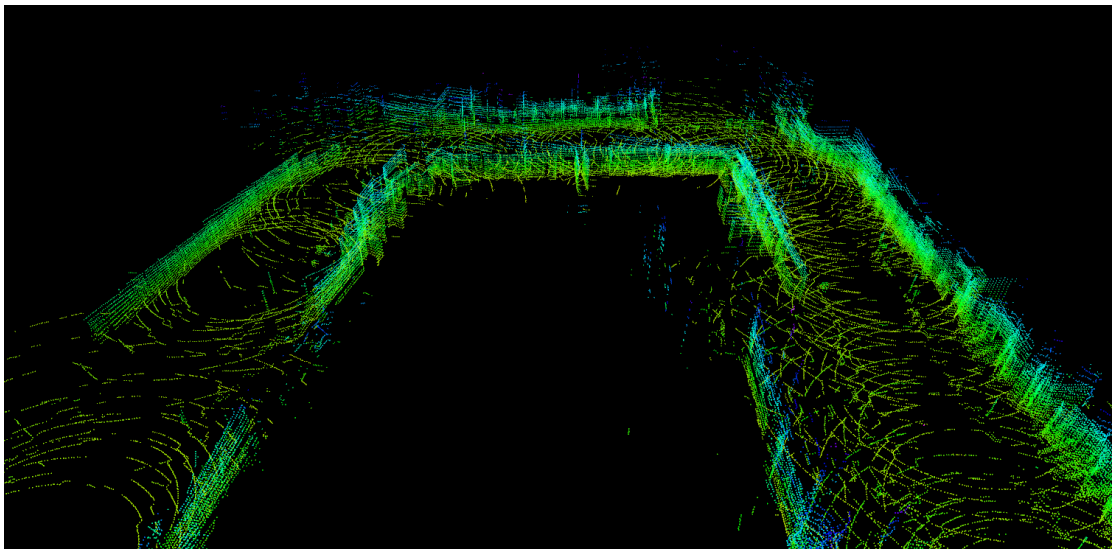

Ilustración 54 Recorrido mundo CARLA aplicando Hdl_Graph_SLAM vista de perspectiva

Capítulo 5.
Resultados.

En este capítulo se van a mostrar los resultados que permiten comparar los diferentes métodos estudiados. En primer lugar, mostrarán los resultados obtenidos en simulación, pero en este caso estableciendo una comparativa cualitativa y cuantitativa de su funcionamiento, utilizando el *Ground Truth* que proporciona el simulador. Después se van a mostrar los resultados obtenidos con los datos reales del vehículo Techs4AgeCar.

5.1. Comparativa de los métodos de SLAM utilizando el simulador

A continuación, representarán gráficamente cada uno de los métodos empleados en este TFM en simulación, comparando la posición del sistema de localización del algoritmo con el *Ground Truth* de Carla.

Para conocer cuál es el algoritmo que ofrece mejores resultados se implementa la métrica de la raíz del error cuadrático medio (RMSE) sobre el error de localización, comparando cuantitativamente el desempeño de cada uno de los métodos.

Para conseguir el objetivo descrito en el párrafo anterior, se ha implementado un código en Matlab que ofrece la posibilidad de realizar una lectura en tiempo real de los topics seleccionados por el usuario.

El *Ground truth* de un sistema de posicionamiento son los valores de localización que se toman como verdaderos. CARLA dispone de esta información en el topic /carla/hero/odometry.

Se almacenan los valores de localización producidos por el algoritmo y por los sensores GPS para el posterior análisis visual y cuantitativo.

5.1.1. LOAM SLAM

En el caso de LOAM SLAM el topic /integrated_to_init/ proporcionará la posición estimada por el sistema de SLAM en cada momento del desarrollo del algoritmo.

5.1.1.1. Simulador CARLA

Los resultados en simulación al comparar el topic de la posición del vehículo en LOAM con el *Ground Truth* de CARLA son los siguientes:

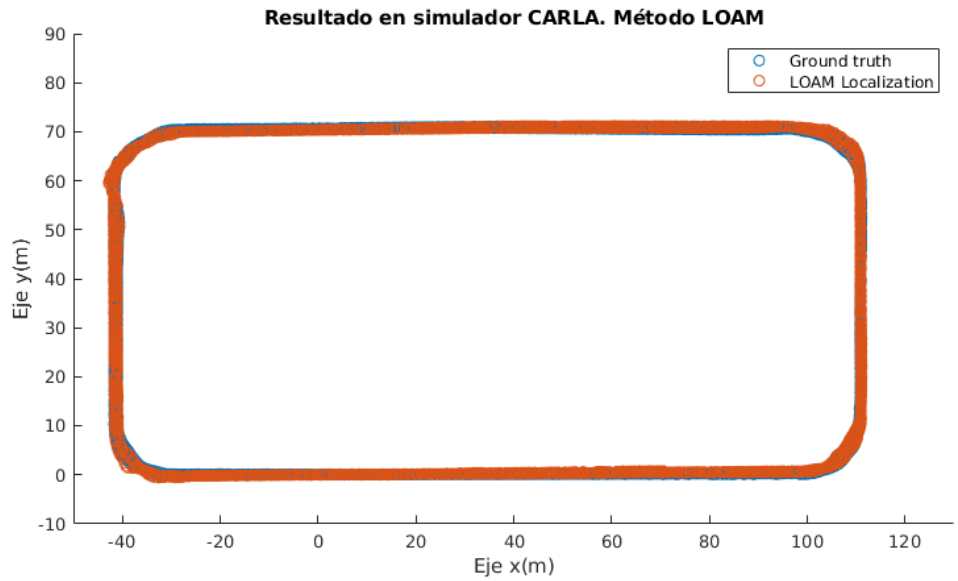

Ilustración 55 Comparativa Ground Truth CARLA VS LOAM Localization

En la Ilustración 55 se ve que los resultados obtenidos por LOAM son muy similares a los del *Ground Truth*, por lo que la ubicación a grandes rasgos es muy exacta.

Ha sido necesario restar la posición inicial tanto al array de posiciones del *Ground Truth* como al de las posiciones del sistema de localización de LOAM debido a que provienen de sistemas de referencia distintos, siendo el *Ground Truth* coordenadas absolutas y LOAM con coordenadas relativas al inicio de la localización.

Para un análisis más preciso se muestra en la Tabla 1 el estudio cuantitativo del error obtenido aplicando la métrica del error cuadrático medio. Se muestra que el error en ambos ejes es inferior al medio metro para un recorrido de aproximadamente medio kilómetro por vuelta, evidenciando una alta precisión.

Error en eje X(m)	0.3074
Error en eje Y(m)	0.3758

Tabla 1 Error cuadrático medio ambos ejes LOAM VS CARLA

Para obtener la distancia entre dos puntos en el espacio, se emplea la distancia Euclídea a partir del teorema de Pitágoras:

$$d_E(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Siendo (x_2,y_2) los puntos obtenidos gracias al sistema de localización del algoritmo y (x_1,y_1) los valores de localización del *Ground Truth*.

A partir del valor obtenido se calcula el valor medio de todas las distancias Euclídeas, obtenido:

$$RMSE\ dE = \sqrt{\frac{\sum d_E}{n}} \quad DME = \frac{\sum \sqrt{d_E}}{n}$$

n: número de puntos

RMSE de la distancia Euclídea (m)	0.3074

Tabla 2 Distancias Euclídeas en simulación LOAM

Es decir, de media un punto obtenido por LOAM a través de su localización estará a 0.3074 metros respecto a ese mismo punto, pero en el sistema de localización del coche.

5.1.2. Hdl_Graph_Slam

En el caso de Hdl_Graph_Slam se han realizado pruebas tanto con el topic /odom como con el topic /hdl_graph_slam/markers para visualizar la posición obtenida a través del sistema de localización de Hdl_Graph_Slam.

5.1.2.1. Simulador CARLA

En primer lugar, se realiza una prueba con el topic /odom de Hdl_Graph_Slam comparándolo con el Ground Truth de CARLA:

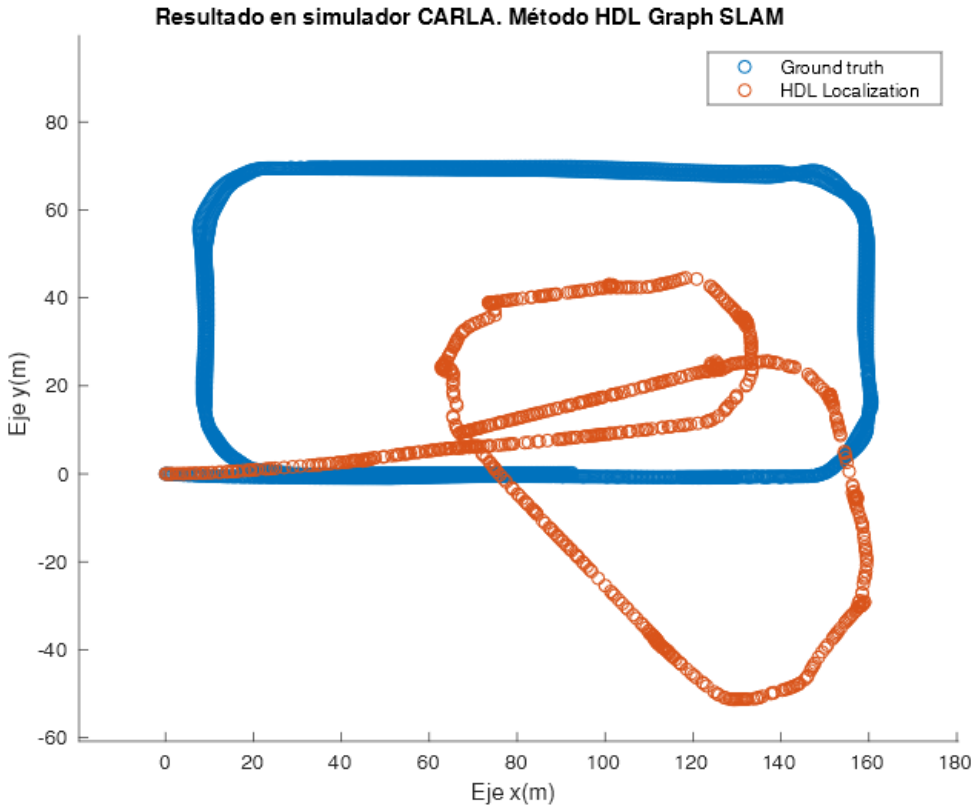

Ilustración 56 Comparativa Ground Truth CARLA VS Hdl Localization

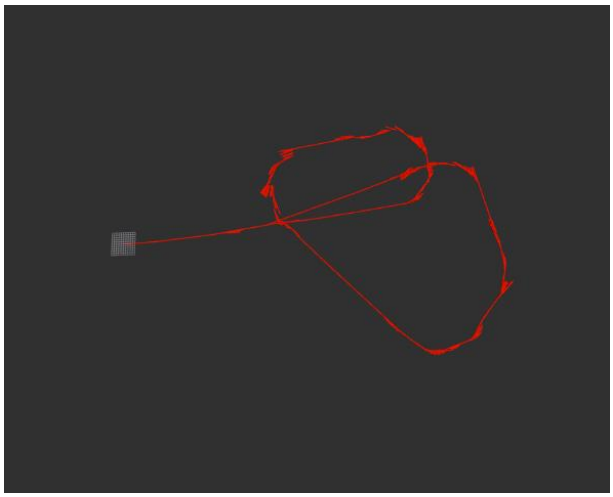

Ilustración 57 Odometría láser Hdl_Graph_SLAM RViz

Como se observa en la Ilustración 56 y la Ilustración 57, Hdl_Graph_SLAM a pesar de obtener un buen mapa, no serviría como sistema de localización online, ya que durante el recorrido tiene muchas desviaciones y tarda en corregirlas, es decir, le cuesta alinear los puntos obtenidos con la odometría laser antes de haber completado una repetición del recorrido.

Después de haber realizado una prueba con el topic /odom se ha decidió comprobar la información del topic /hdl_graph_slam/markers con la esperanza de obtener mejores resultados.

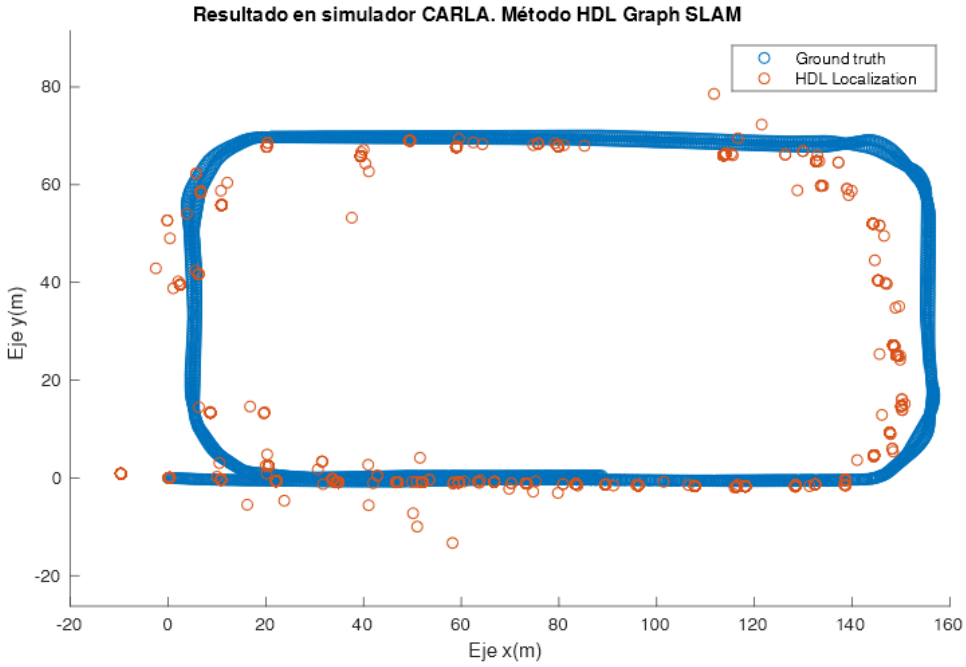

Ilustración 58 Comparativa Ground Truth CARLA VS Hdl_Graph_SLAM Localization

Los puntos rojos que corresponden con localización de Hdl_Graph_SLAM que se muestran en la Ilustración 58 aparecen con relativa exactitud después de haber hecho todo el recorrido, pero durante el mismo, existe mucha desalineación de la odometría.

Ha sido necesario restar la posición inicial del array de posiciones del *Ground Truth,* pero en el caso de las posiciones del sistema de localización de Hdl_Graph_SLAM no lo ha sido, al empezar el recorrido en la posición (0,0).

Aun así, para un análisis más preciso se muestra en la Tabla 3 el estudio cuantitativo del error obtenido aplicando la métrica del error cuadrático medio. Se muestra que el error en ambos ejes es inferior al medio metro para un recorrido de aproximadamente medio kilómetro por vuelta, evidenciando una alta precisión.

Error en eje X(m)	4.36
Error en eje Y(m)	0.6122

Tabla 3 Error cuadrático medio ambos ejes Hdl_Graph_SLAM VS CARLA

RMSE de la distancia Euclídea (m)	**4.40**

Tabla 4 Distancias Euclídeas en simulación LOAM

Es decir, de media un punto obtenido por Hdl_Graph_SLAM a través de su localización estará a 4.40 metros respecto a ese mismo punto, pero en el sistema de localización del coche.

Por lo tanto, el método Hdl_Graph_SLAM se descarta como algoritmo para poder ser implementado en el vehículo real, al no ser válido para la navegación autónoma pues posee una odometría con un elevado nivel de ruido y severas discontinuidades y derivas.

5.2. Resultados con el vehículo autónomo en entorno real

Se realiza el mismo proceso aplicado en el simulador CARLA, pero esta vez en un entorno no simulado, más concretamente el Campus Externo de la Universidad de Alcalá, en la Calle Espada.

Se selecciona esta calle para hacer la prueba real con el vehículo autónomo, debido a los incesantes problemas que genera el GPS en la localización del coche producidas por las pérdidas de información del sensor por los árboles y en la odometría por las irregularidades en el firme de la propia calle.

Del mismo modo que en CARLA, se genera un recorrido de dos vueltas a las dos rotondas que conforman la calle para comprobar el correcto funcionamiento del cierre de lazo del sistema.

Ilustración 59 Recorrido Real Vehículo Autónomo, LOAM SLAM

5.2.1. Google Cartographer

Con Google Cartographer no ha sido posible realizar un recorrido con el vehículo autónomo en un entorno real debido a la necesidad por parte del algoritmo de obtener datos a través de una IMU, sensor que no posee el coche del proyecto Techs4AgeCar.

5.2.2. LOAM SLAM

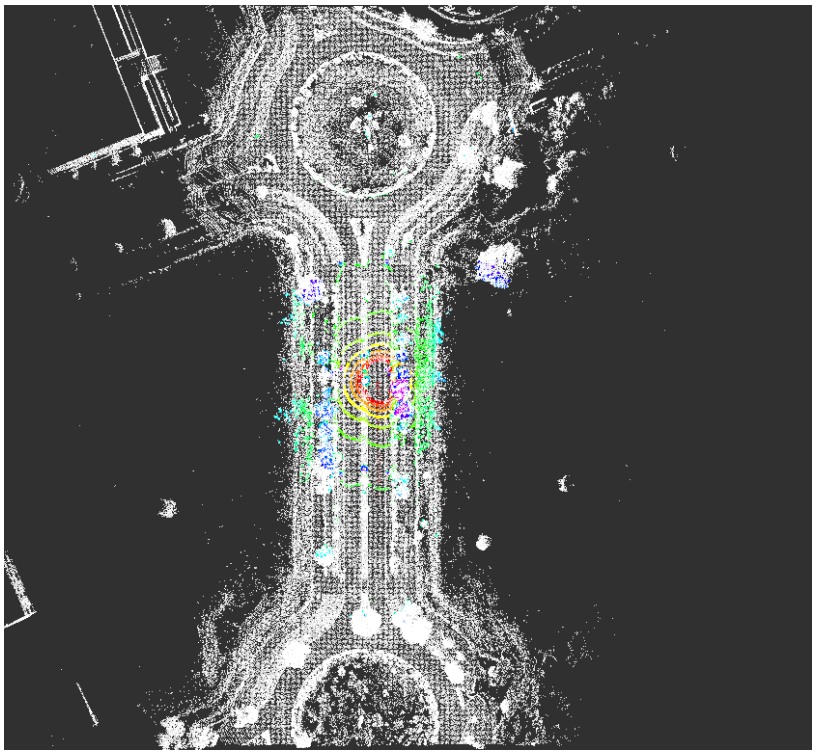

Ilustración 60 Recorrido entorno real aplicando LOAM SLAM

Empleando el método LOAM, al igual que en CARLA, los resultados son excelentes. El error acumulado consigue compensarse, de manera que no afecte a la hora de unir las nubes de puntos para generar el mapa.

Los árboles o incluso la mediana de la calle pueden identificarse con claridad, lo que aporta una solución al problema de la pérdida de información del sensor GPS.

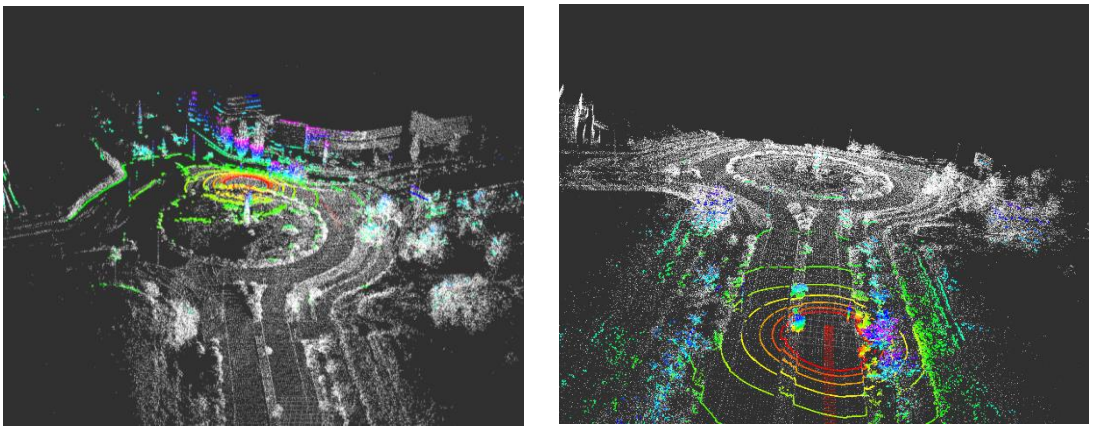

Ilustración 61 Recorrido entorno real vista en perspectiva, LOAM SLAM

5.2.2.1. Análisis cualitativo

Los resultados en Matlab al comparar el topic de la posición del vehículo en LOAM con la localización actual del coche real son los siguientes:

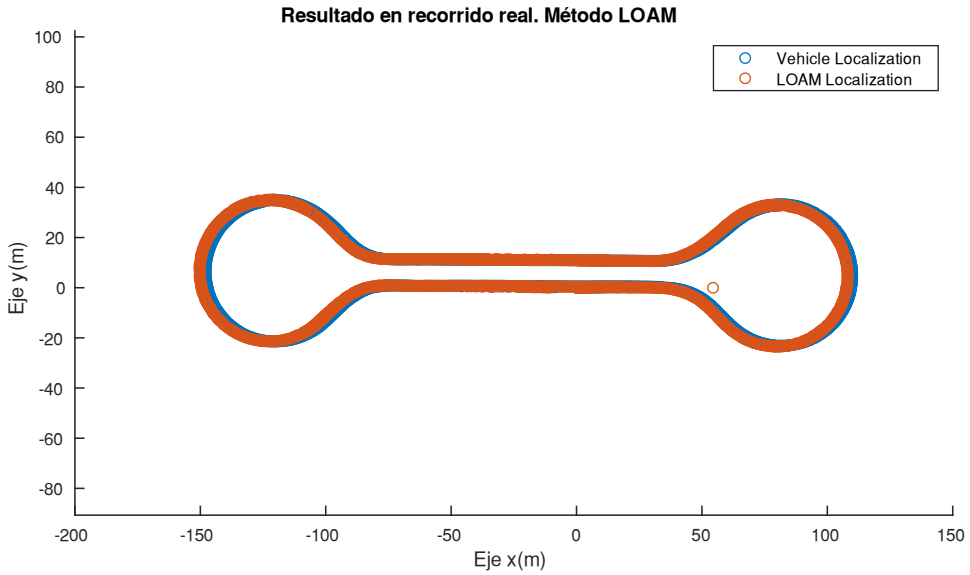

Ilustración 62 Comparativa localización actual vehículo real VS LOAM Localization

Ha sido necesario restar la posición inicial tanto al array de posiciones del sistema de localización del vehículo real, como a las posiciones del sistema de localización de LOAM, ya que provienen de sistemas de referencia distintos, siendo el del vehículo coordenadas absolutas (odometría de encoders con aportación de GPS) y LOAM con coordenadas relativas al inicio de la localización

Al igual que ocurre en CARLA, el algoritmo LOAM ofrece buenos resultados en entorno real.

Hay que destacar que el sistema de localización que ofrece LOAM es mucho más exacto que el que había antes implementado en el vehículo de Techs4AgeCar, ya que se ayuda de agentes externos como árboles, farolas o incluso aceras para obtener referencias que permitan enlazar las progresivas nubes de puntos obtenidas por el sensor laser, conociendo con precisión los desplazamientos producidos.

5.2.3. Hdl_Graph_SLAM

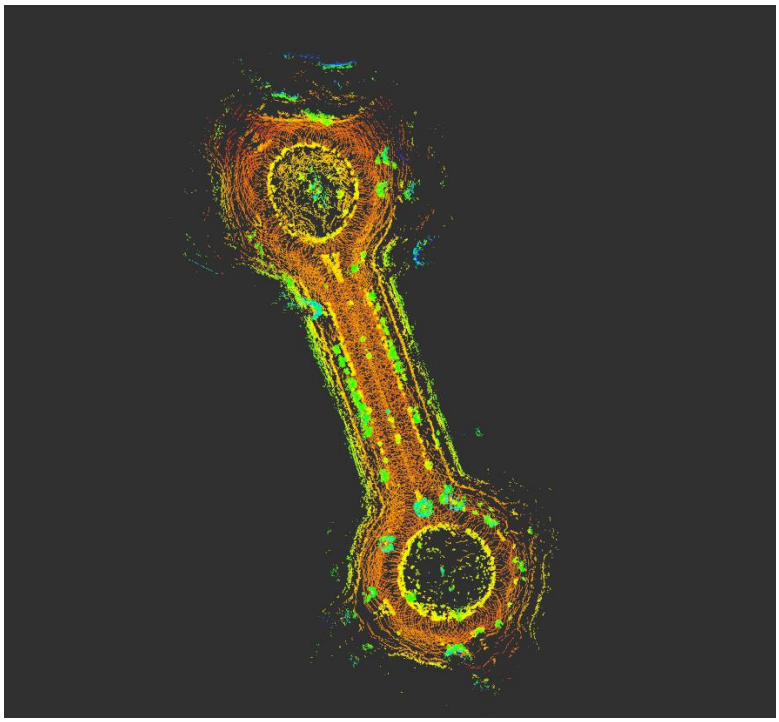

Ilustración 63 Recorrido en entorno real aplicando Hdl_Graph_SLAM

Hdl_Graph_SLAM genera un mapa del entorno muy semejante al mapa real obtenido a través de Google Maps.

Las restricciones del GPS funcionan bastante bien, consiguiendo eliminar los errores acumulados que se pueden dar por la coincidencia de escaneos.

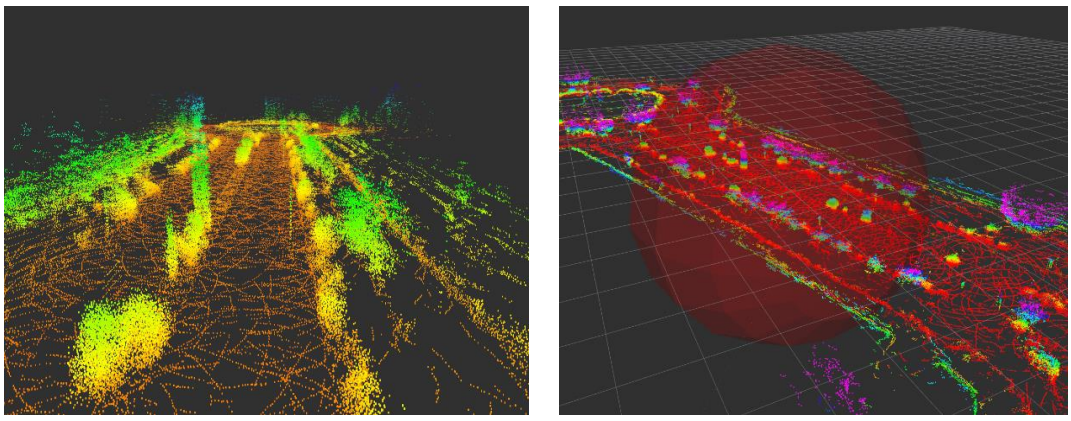

Ilustración 64 Recorrido entorno real vista en perspectiva, Hdl_Graph_SLAM

El algoritmo consigue reconstruir el mapa con alto nivel de detalle, pudiendo observar la vegetación del entorno o las líneas que dividen las calles.

5.3. Método seleccionado

Finalmente, al descartar Google Cartographer y Hdl_Graph_SLAM, LOAM será el método seleccionado para ser implementado en el vehículo real y mejorar su sistema de mapeado y localización.

5.3.1. Odometría LOAM VS Odometría Encoders

Sabiendo cuál es el algoritmo seleccionado, se debe comprobar realmente si la odometría generada por el algoritmo LOAM es mejor que el sistema de localización que tiene ahora mismo el vehículo real implementado.

Actualmente la localización del vehículo emplea información GPS fusionada con la odometría proporcionada por dos encoders ubicados en las ruedas traseras, de forma que cada vez que el sistema recibe una ubicación GPS precisa reinicia la odometría, mientras que cuando el vehículo es incapaz de conocer su posición GNSS por pérdida de contacto con los satélites, la odometría proporciona soporte para el mantenimiento del vehículo en la trayectoria.

Para comprobar el desempeño de la odometría proporcionada por el algoritmo LOAM, es necesario desactivar la recepción de medidas por parte del sensor GPS y así adquirir únicamente la odometría de los encoders con su error incremental.

La odometría de los encoders sin la influencia del GPS se publica en el topic /Localization/odometry_pose, mientras que, la odometría lidar se guarda en el topic /integrated_to_init explicado en anteriores capítulos.

La Ilustración 65 muestra el resultado de la prueba descrita:

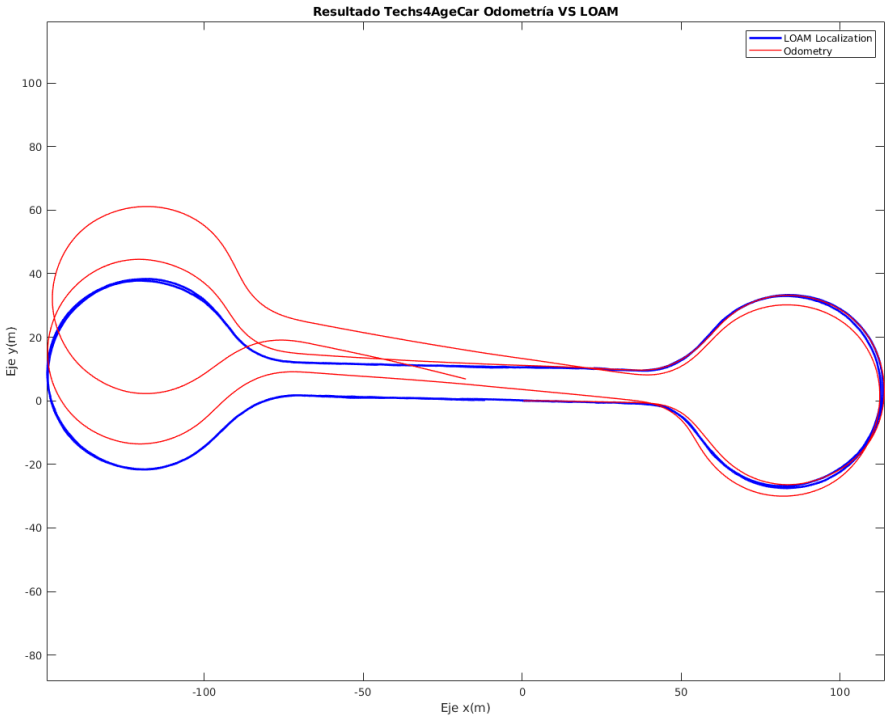

Ilustración 65 Resultado Techs4AgeCar Odometría Encoders VS Odometría LOAM

La odometría de los encoders presenta derivas incrementales por lo que este sistema es incapaz de producir un cierre de lazo en el recorrido. Sin embargo, la odometría generada por el algoritmo LOAM se surte del propio mapa para corregir el error acumulado, realizando un cierre de lazo con elevada precisión.

5.3.2. Fusión con GPS en filtro de Kalman extendido

Finalmente se decide sustituir la entrada de odometría del filtro EKF, que en el sistema de localización actual es la odometría de los encoders más la posición del sensor GPS, con la odometría lidar que devuelve el algoritmo LOAM.

Esta prueba permite observar si realmente la posición que se obtiene en la salida del EKF mejora introduciendo la odometría lidar. La Ilustración 66 muestra cuáles son las entradas y salidas del filtro de Kalman del vehículo real si se implementa el método LOAM.

Ilustración 66 Esquema Filtro EKF con LOAM

El filtro de Kalman requiere que la odometría lidar y la posición GPS se encuentren en el mismo sistema de referencia ya que originalmente la odometría se genera un sistema local mientras que la localización GPS presenta valores absolutos.

Por lo tanto, para introducir los datos en el filtro es necesario aplicar una matriz de transformación entre ambos sistemas de coordenadas, de forma que ambos sistemas de coordenadas se encuentren superpuestos.

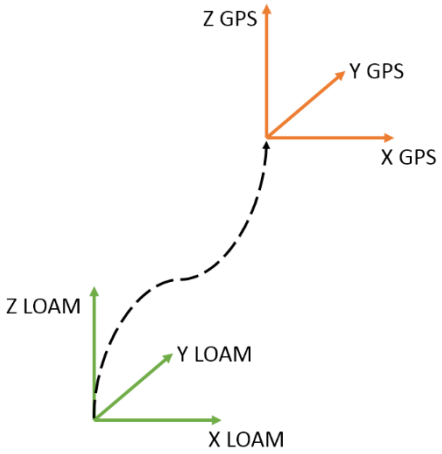

Ilustración 67 Transformación de los sistemas de coordenadas

La Ilustración 68 muestra la superposición de las entradas GPS y odometría LOAM al filtro EKF, junto con la salida ponderada.

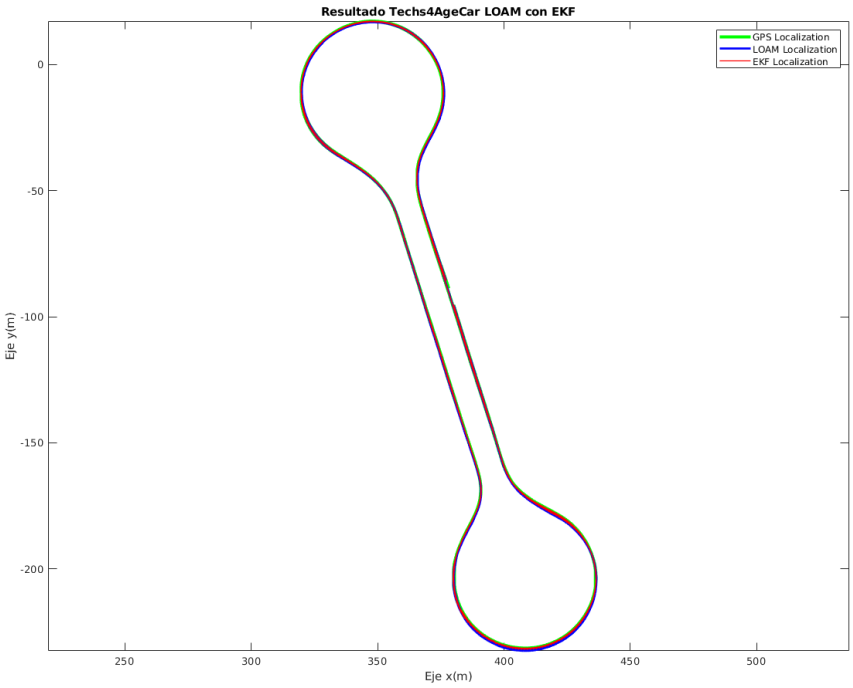

Ilustración 68 Resultado Techs4AgeCar LOAM con EKF

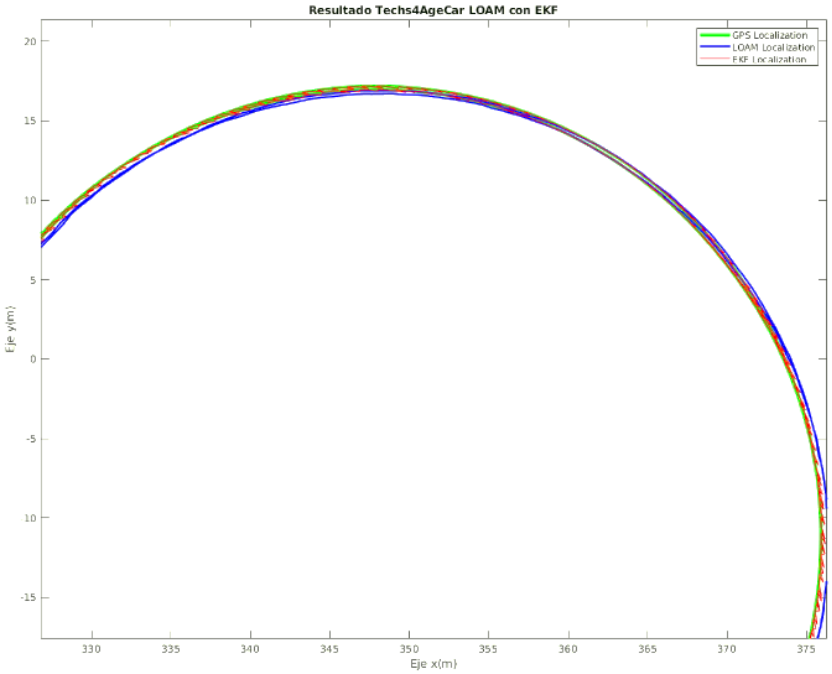

Ilustración 69 Zoom Resultado Techs4AgeCar LOAM con EKF

En base a las ilustraciones anteriores las localizaciones GPS y odometría lidar presentan una elevada coincidencia debido a la elevada precisión de esta última. La salida proporcionada por el filtro EKF presenta valores intermedios entre ambas.

Con el sistema de localización anterior, existen situaciones en las que la información del GPS se pierde en algún momento del recorrido con el coche real, y este emplea la odometría de los encoders para poder localizarse y situarse en el mapa, produciéndose fuertes derivas al ser no ser un método muy eficaz. Sin embargo, con la sustitución de esta odometría por una basada en LIDAR se eliminarán estos errores.

Capítulo 6.
Conclusiones y trabajos futuros.

A lo largo del desarrollo de este Trabajo Fin de Máster se han evaluado distintas técnicas del estado del arte con el objetivo de implementar aquel sistema de localización y mapeado que produzca los mejores resultados tanto en simulación como en entorno real.

En primer lugar, se ha estudiado el método Google Cartographer, que ofrecía una solución al problema del SLAM basada en láser, IMU e información de odometría. Tras la realización de pruebas en simulación se detectaron problemas como la producción de un mapa en dos dimensiones o el fallo en cierre de lazo, que junto con la falta de sensor IMU por parte del vehículo real, han sido determinantes para la eliminación de este algoritmo como posible método a implementar en el vehículo del proyecto Techs4AgeCar.

En segundo lugar, se procedió al estudio del algoritmo Hdl_Graph_SLAM, el cual necesita de la información de la nube de puntos del láser, así como los datos de posición GPS para poder solucionar el problema del SLAM. Al realizar las pruebas en simulación se llegó a la conclusión de que no serviría como sistema de localización online, ya que durante el recorrido presenta elevadas desviaciones y tarda en corregirlas, es decir, le cuesta alinear los puntos obtenidos con la odometría laser antes de haber completado una repetición del recorrido. Finalmente, al no servir como método en tiempo real y debido al elevado valor de error RMSE demasiado elevado, se decidió no llegar a la etapa de implementación con este método.

Por último, se estudia el método LOAM SLAM, que requiere únicamente de la nube de puntos para poder generar tanto el mapa incremental como la odometría necesaria para resolver el problema de localización y mapeado en tiempo real.

En la fase de pruebas sobre el simulador CARLA se obtuvieron muy buenos resultados tanto en el análisis visual como en el análisis cuantitativo con el cálculo del error RMSE, siendo este bastante más bajo que el resultante del método Hdl_Graph_SLAM.

Debido a los buenos resultados se realizó una comparativa de la salida de LOAM con la odometría del sistema de localización actual del vehículo real.

Hay que destacar que el sistema de localización que ofrece LOAM presenta una precisión mucho mayor respecto al sistema de odometría previo, ya que se ayuda de agentes externos como árboles, farolas o incluso aceras para obtener referencias que permitan enlazar las progresivas nubes de puntos obtenidas por el sensor láser, conociendo con precisión los desplazamientos producidos.

La odometría de los encoders del sistema actual presenta derivas incrementales por lo que este sistema es incapaz de producir un cierre de lazo en el recorrido. Sin embargo, la odometría generada por el algoritmo LOAM se surte del propio mapa para corregir el error acumulado, realizando un cierre de lazo con elevada precisión.

Posteriormente se implementa el método LOAM en el filtro de Kalman del vehículo Techs4AgeCar. Los resultados obtenidos muestran que las localizaciones GPS y odometría lidar presentan una elevada coincidencia debido a la alta precisión de esta última. La salida proporcionada por el filtro EKF produce valores intermedios entre ambas.

Con el sistema de localización anterior, existen situaciones en las que la información del GPS se pierde en algún momento del recorrido con el coche real, y este emplea la odometría de los encoders para poder localizarse y situarse en el mapa, produciéndose fuertes derivas al ser no ser un método muy eficaz. Sin embargo, con la sustitución de esta odometría por una basada en LIDAR se eliminarán estos errores.

Como trabajos futuros se propone la creación de un mapa completo en el entorno del Campus Externo de la Universidad de Alcalá y su posterior fusión con el sistema de mapeado actual basado en lanelets, de forma que se produzca un mapeado coordinando la localización del algoritmo LOAM con la planificación de trayectorias.

Anexo I. Manual de usuario

En este anexo se indican los pasos necesarios para instalar y ejecutar los diferentes métodos y entornos necesarios para llevar a cabo la reproducción de los resultados presentes en este Trabajo Fin de Máster.

I.I Simulador Carla

La versión de Carla empleada en este TFM ha sido la 0.9.7, una versión más que consolidada. Principalmente se ha escogido esta versión por que entre sus múltiples sensores disponibles incluye IMU, instrumento indispensable en alguno de los métodos propuestos en este trabajo.

Para instalar cualquier versión estable es necesario seguir los pasos especificados en esta página web: https://carla.readthedocs.io/en/latest/build_linux/ .

Al ser necesario realizar una lectura de los diferentes sensores para obtener la información del vehículo autónomo, se debe instalar el paquete de ROS diseñado específicamente para dicha tarea: https://github.com/carla-simulator/ros-bridge

Comandos para la instalación de CARLA Y CARLA ROS bridge

- CARLA

```
sudo apt-get install aria2
git clone https://github.com/carla-simulator/carla
cd ~/carla
./Update.sh
echo 'export UE4_ROOT=~/UnrealEngine_4.24' >> ~/.bashrc
make launch
make PythonAPI
make package
echo 'export PYTHONPATH=$PYTHONPATH:~/carla/PythonAPI/carla/dist/
carla-<carla_version_and_arch>.egg' >> ~/.bashrc
```

- CARLA ROS bridge

```
sudo apt-get install aria2
git clone https://github.com/carla-simulator/carla
cd ~/carla
./Update.sh
echo ‘export UE4_ROOT=~/UnrealEngine_4.24’ >> ~/.bashrc
make launch
make PythonAPI
make package
echo ‘export PYTHONPATH=$PYTHONPATH:~/carla/PythonAPI/carla/dist/
carla-<carla_version_and_arch>.egg’ >> ~/.bashrc
```

Comandos para la ejecución de CARLA

- Lanzar servidor

```
cd ~/carla
./CarlaUE4.sh#Puede estar en algún subdirectorio
```

- Lanzar cliente

```
cd ~/carla/PythonAPI/examples
python manual_control.py
```

- Abrir CARLA ROS bridge

```
roslaunch carla_ros_bridge carla_ros_bridge.launch
```

- Lanzar múltiples vehículos circulando autónomamente

```
cd ~/carla/PythonAPI/examples
python spawn_npc.py
```

- Cambiar ciudad de simulación a Town02 (o cualquier otra)

```
cd ~/carla/PythonAPI/utils
python config.py -m Town02
```

I.II Google Cartographer

Google Cartographer dispone de una página web con una gran cantidad de información acerca del algoritmo, como instalarlo como ejecutarlo etc.

Como instalar Cartographer

- Instalación herramientas

```
sudo apt-get update
sudo apt-get install -y python3-wstool python3-rosdep ninja-build
sudo apt-get update
sudo apt-get install -y python-wstool python-rosdep ninja-build
```

- Creación de un nuevo workspace en 'catkin_ws'

```
mkdir catkin_ws
cd catkin_ws
wstool init src
wstool merge -t src https://raw.githubusercontent.com/cartographer-
project/cartographer_ros/master/cartographer_ros.rosinstall
wstool update -t src
```

- Instalar dependencias de 'cartographer_ros'

```
src/cartographer/scripts/install_proto3.sh
sudo rosdep init
rosdep update
rosdep install --from-paths src --ignore-src --rosdistro=${ROS_DISTRO}
-y
```

- Construir e Instalar

```
catkin_make_isolated --install --use-ninja
```

Ejecutar Cartographer con tu propio *Rosbag*

- Validar tu *rosbag* en el entorno de Cartographer ROS

```
cartographer_rosbag_validate -bag_filename your_bag.bag
```

- Ejecutar tu *rosbag* con Cartographer

```
roslaunch cartographer_ros my_robot.launch
bag_filename:=/path/to/your_bag.bag
```

I.III LOAM SLAM

Los comandos de instalación y ejecución de LOAM se encuentran disponible en un GitHub, que contiene además información sobre el algoritmo.

Como instalar LOAM SLAM

- Construir LOAM con *catkin*

```
cd ~/catkin_ws/src/
git clone https://github.com/laboshinl/loam_velodyne.git
cd ~/catkin_ws
catkin_make -DCMAKE_BUILD_TYPE=Release
source ~/catkin_ws/devel/setup.bash
```

Como ejecutar LOAM SLAM

- Ejecutar LOAM en un terminal

```
cd ~/catkin_ws/src/
git clone https://github.com/laboshinl/loam_velodyne.git
cd ~/catkin_ws
catkin_make -DCMAKE_BUILD_TYPE=Release
source ~/catkin_ws/devel/setup.bash
```

- En un Segundo terminal ejecutar el rosbag

```
rosbag play ~/Downloads/velodyne.bag
```

I.IV Hdl_Graph_SLAM

Al igual que LOAM SLAM, Hdl_Graph_SLAM posee un GitHub con información sobre los pasos para su instalación y ejecución, así como unos cuantos ejemplos que pueden servir como entrenamiento al usuario.

Paquetes previa instalación

- **Librerías:**
  - OpenMP
  - PCL 1.7
  - g2o

```
cd g2o
git checkout a48ff8c42136f18fbe215b02bfeca48fa0c67507
```

- suitesparse

```
sudo apt-get install libsuitesparse-dev
git clone https://github.com/RainerKuemmerle/g2o.git
```

- **Paquetes necesarios para ROS**
  - Geodesy :

```
# for indigo
sudo apt-get install ros-indigo-geodesy ros-indigo-pcl_ros ros-indigo-
nmea-msgs
# for kinetic
sudo apt-get install ros-kinetic-geodesy ros-kinetic-pcl-ros ros-
kinetic-nmea-msgs ros-kinetic-libg2o
# for melodic
sudo apt-get install ros-melodic-geodesy ros-melodic-pcl-ros ros-
melodic-nmea-msgs ros-melodic-libg2o


cd catkin_ws/src
git clone https://github.com/koide3/ndt_omp.git
```

  - nmea_msgs
  - pcl_ros
  - ndt_omp

Para instalar finalmente Hdl_Graph_SLAM

```
mkdir build && cd build
cmake .. -DCMAKE_BUILD_TYPE=RELEASE
make -j8
sudo make install
```

En algunas ocasiones aparece un error de memoria, que se puede solucionar instalando Hdl_Graph_SLAM en el entorno Docker como último recurso:

```
roscd hdl_graph_slam
sudo docker build --tag hdl_graph_slam -f docker/kinetic/Dockerfile .
```

Finalmente, para ejecutar el algoritmo:

```
sudo docker run -it --net=host --rm hdl_graph_slam bash
source /root/catkin_ws/devel/setup.bash
roslaunch hdl_graph_slam hdl_graph_slam.launch
```

Anexo II. Presupuesto

En este anexo se detalla el total de los costes estimados para la realización del proyecto en los aspectos de mano de obra, así como hardware y software necesario.

II.I Coste Hardware

Hardware	Precio unitario	Unidades	Precio total
Ordenador sobremesa	485 €	1	485 €
Nvidia GEFORCE GT 430	80 €	1	80 €
SSD Samsung 250 GB	50 €	1	50 €
i5-2500	155 €	1	155 €
Otros componentes	200 €	1	200 €
Costes HW totales			970 €

Tabla 5 Costes de material Hardware

II.II Coste Software

En la realización del trabajo se ha utilizado diverso material software. Los costes producidos por este material no suponen ningún coste al usuario debido a las licencias que posee la Universidad de Alcalá para los estudiantes.

Software	Precio
Ubuntu 16.04 LTS	- €
CARLA	- €
Python	- €
Microsoft Office	- €
Matlab licencia Universitaria	- €
Costes SW	- €

Tabla 6 Coste Software

II.III Costes de personal

El coste de personal de este Trabajo Fin de Máster se estima a partir de un sueldo medio de 18 €/hora de un Ingeniero Industrial, trabajando una media de 4 horas diarias.

	Meses	Días	Precio
Estudio	1	22	1.584 €
Análisis y mejora	1	22	1.584 €
Descarga e instalación	2	44	3.168 €
Estudio CARLA	1	22	1.584 €
Adaptación método	2	44	3.168 €
Pruebas	1	22	1.584 €
Corrección	1	22	1.584 €
Redacción	2	44	3.168 €
Costes de personal			17.424 €

Tabla 7 Costes de personal

II.IV Costes de ejecución totales

Los costes de ejecución del proyecto son:

Costes de ejecución	
Costes Hardware	970,0 €
Costes Software	0,0 €
Costes de personal	17.424,0 €
Subtotal final	18.394,0 €

Tabla 8 Costes de ejecución totales

II.V Gastos generales y beneficio industrial

Según el PEM 2020, los gastos generales de un proyecto se estiman en un 13% de los costes de ejecución totales. El beneficio industrial se estima entorno al 6% de los costes de ejecución totales.

Concepto	Subtotal
Costes de ejecución material	18.394,00 €
Gastos generales (13%)	2.391,22 €
Beneficio industrial (6%)	1.103,64 €
Subtotal final	21.888,86 €

Tabla 9 Gastos generales y beneficio industrial

II.VI Importe total del presupuesto

El importe total de este Trabajo Fin de Máster se calcula incluyendo el IVA al presupuesto de ejecución por contrata. Este impuesto se sitúa en un 21% del valor.

Concepto	
Presupuesto de ejecución	21.888,9 €
Porcentaje de IVA	4.596,66 €
Importe final	26.485,52 €

Tabla 10 Importe total del presupuesto

El presupuesto final estimado de este Trabajo Fin de Grado asciende a un total de 26.485,52 € (veintiséis mil cuatrocientos ochenta y cinco con cincuenta y dos euros).

Anexo III. Pliego de condiciones

A continuación, se detalla el total del material empleado para la realización de este Trabajo Fin de Máster:

III.I Software empleado

Durante el desarrollo de este trabajo se han utilizado los programas que se detallan en la siguiente lista, todos ellos bajo el sistema operativo Windows 10:

- Matlab R2018b.
- Python 2.
- Carla 0.9.7.
- ROS Kinetic.
- Ubuntu 16.04 LTS.
- Microsoft Word 2016.

III.II Hardware empleado

El hardware utilizado para el desarrollo del trabajo se encuentra especificado en la siguiente lista:

- Ordenador sobremesa, con los siguientes componentes:
    - Nvidia GEFORCE GT 430.
    - SSD Samsung 250 GB.
    - i5-2500.

Bibliografía

[1] «Robesafe,» [En línea]. Available: https://www.robesafe.uah.es/index.php/en/.

[2] «Velodyne Lidar,» [En línea]. Available: https://velodynelidar.com/.

[3] «ROS,» [En línea]. Available: https://www.ros.org/.

[4] «Localización y mapeado simultáneos,» [En línea]. Available: https://es.wikipedia.org/wiki/Localizaci%C3%B3n_y_modelado_simult%C3%A1neos.

[5] S. Thrun, Probabilistic Robotics.

[6] G. Bresson, Z. Alsayed, L. Yu y S. Glaser, «Simultaneous Localization And Mapping: A Survey of Current Trends in Autonomous Driving,» 2017.

[7] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel y M. Werling, «Towards Fully Autonomous Driviing: Systems and Algorithms,» *IEE Intelligent Vehicles Symposium,* 2011.

[8] F. Kunz, D. Nuss, J. Wiest, H. Deusch, S. Reuter y F. Gritschneder, «Autonomous Driving at Ulm University: A Modular, Robust, and Sensor-Independent Fusion Approach,» *IEEE Intelligent Vehicles Symposium,* 2015.

[9] M. Aeberhard, S. Rauch, M. Bahram y G. Tanzmeister, «Experience,Results and Lessons Learned from Automated Driving on Germanys Highways,» *IEEE Intelligent Transportation Systems Magazine,* 2015.

[10] «MRPT,» [En línea]. Available: https://www.mrpt.org/list-of-mrpt-apps/application-pf-localization/.

[11] E. Rodríguez, J. Blanco, J. Torres, J. Moreno, A. Giménez y J. Guzmán, «A ROS REACTIVE NAVIGATION SYSTEM FOR GROUND VEHICLES BASED ON TP-SPACE TRANSFORMATIONS,» 2017.

[12] M. A. Rueda-Castro, J. Monroy, F.-Á. Moreno y J. Gonzalez-Jimenez, «Integración de un Planificador de Trayectorias Parametrizado en la Arquitectura Robótica ROS,» 2016.

[13] G. Canberk Suat, «REAL-TIME 2D AND 3D SLAM USING RTAB-MAP, GMAPPING AND CARTOGRAPHER PACKAGES,» pp. https://www.researchgate.net/publication/326986124_REAL-TIME_2D_AND_3D_SLAM_USING_RTAB-MAP_GMAPPING_AND_CARTOGRAPHER_PACKAGES, 2018.

[14] «Github Gmapping SLAM,» [En línea]. Available: https://github.com/Project-MANAS/slam_gmapping.

[15] P. Abbeel, «Rao-Blackwellized Particle Filtering,» *People.eecs.berkeley.*

[16] E. H. Chouaib Harik y A. Korsaeth, «Combining Hector SLAM and Artificial Potential Field for Autonomous Navigation Inside a Greenhouse,» 2018.

[17] S. Kohlbrecher, J. Meyer, T. Graber, K. Petersen, U. Klingauf y O. v. Stryk, «Hector Open Source Modules for Autonomous Mapping and Navigation with Rescue Robots,» 2014.

[18] «Github MCL_3dl,» [En línea]. Available: https://github.com/at-wat/mcl_3dl.

[19] «Cartographer-ros.readthedocs.io,» [En línea]. Available: https://google-cartographer-ros.readthedocs.io/en/latest/index.html.

[20] A. Dosovitskiy, G. Ros, F. Codevilla, A. López y V. Koltun, «CARLA: An Open Urban Driving Simulator,» *1st Conference on Robot Learning, Mountain View,* 2017.

[21] R. Barea Navarro, E. J. Molinos Vicente, M. Miguel Ocaña y M. Ocaña Miguel, «Introducción a a ROS,» de *Robótica Móvil y Percepcción*, Universidad de Alcalá.

[22] «Carla.readthedocs.io,» [En línea]. Available: https://carla.readthedocs.io/en/latest/ros_installation/.

[23] W. Hess, D. Kohler, H. Rapp y D. Andor, «Real-Time Loop Closure in 2D LIDAR SLAM,» 2016.

[24] J. Zhang y S. Singh, LOAM: Lidar Odometry and Mapping in Real-time, 2014.

[25] «Punto más cercano iterativo (ICP),» [En línea]. Available: https://es.qwe.wiki/wiki/Iterative_closest_point#:~:text=ICP%20es%20uno%20de%20los,Medioni%20y%20Besl%20y%20McKay..

[26] «Github Hdl_Graph_SLAM,» [En línea]. Available: https://github.com/koide3/hdl_graph_slam.

[27] J. Miura, K. Koide y E. Menegatti, «A portable three-dimensional LIDAR-based system for long-term and wide-area people behaviour measurement,» 2019.

[28] M. A. Fischler y R. C. Bolles, «Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography,» 1981.

[29] C. Gómez Huélamo, Predictive Techniques for Scene Understanding by using Deep Learning, 2019.

[30] M. Ocaña Miguel y E. López Guillen, Introducción a la Robótica Móvil.

[31] «Gps Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/GPS.

[32] J. F. Arango Vargas, Trabajo Fin de Máster: Diseño de un sistema drive-by-Wire para un vehículo autónomo, 2020.

[33] «ROS.org: robot_localization,» [En línea]. Available: http://wiki.ros.org/robot_localization.

[34] T. Pire, T. Fischer, J. Cierva, P. De Cristóforis y J. J. Berlles, «Stereo Parallel Tracking and Mapping for robot localization».

[35] «ROS.org mrpt_localization,» [En línea]. Available: http://wiki.ros.org/mrpt_localization.

[36] «CARLA ROS-Bridge GitHub,» [En línea]. Available: https://github.com/carla-simulator/ros-bridge.

Universidad de Alcalá

Escuela Politécnica Superior

ESCUELA POLITECNICA SUPERIOR

Universidad de Alcalá